E

eAccessibility

Constantine Stephanidis[1,2]
[1]Foundation for Research and Technology – Hellas (FORTH), Heraklion, Greece
[2]University of Crete, Heraklion, Greece

Definition

eAccessibility refers to the access of Information and Communication Technologies (ICT) by people with disabilities, with particular emphasis on the World Wide Web. It is the extent to which the use of an application or service is affected by the user's particular functional limitations or abilities (permanent or temporary). eAccessibility can be considered as a fundamental prerequisite of usability.

Historical Background

The percentage of disabled citizens has increased dramatically in the last century due to life condition improvements, higher life expectancy, and population aging. This trend is anticipated to further increase in the next decades.

Traditional efforts to provide eAccessibility for users with disabilities were based on the adaptation of applications and services originally developed for able-bodied users. In the context of the Information Society, this raises the fundamental issue of granting to disabled citizens access to a variety of technologies that become progressively more entangled with all types of everyday activities.

Early technical approaches to eAccessibility mainly focused on two directions. The first treats each application separately, and takes all the necessary implementation steps to arrive at an alternative accessible version (*product-level adaptation)*. Practically, product-level adaptation often implies redevelopment from scratch. The second approach "intervenes" at the level of the particular interactive application environment (e.g., MS-Windows) in order to provide appropriate software and hardware technology so as to make that environment accessible through alternative means (*environment-level adaptation*). The latter option extends the scope of eAccessibility to cover potentially all applications running under the same interactive environment, rather than a single application, and is therefore acknowledged as a more promising strategy.

The above approaches have given rise to several methods for addressing eAccessibility, including techniques for the configuration of input/output at the level of the user interface, and the provision of Assistive Technologies. Popular Assistive Technologies supporting eAccessibility include screen readers and Braille displays for blind users, screen magnifiers for users with low vision, alternative input and output devices for motor impaired users (e.g., adapted keyboards, mouse emulators, joystick, binary switches), specialized browsers (e.g., [10]), and text prediction systems (e.g., [4]). Assistive Technologies are legally defined in the US as "Any item, piece of equipment, or system, whether acquired commercially, modified, or customized, that is commonly used to increase, maintain, or improve functional capabilities of individuals with disabilities [15]."

Despite progress, Assistive Technologies and dedicated design approaches have been criticized for their essentially reactive nature [13]. Although the "reactive" approach to eAccessibility may be the only viable solution in many cases, it suffers from potentially serious shortcomings, such as limited and low quality access, as well as difficulties in development, maintenance, and keeping pace with technological evolution.

Therefore, the need for more systematic and proactive approaches to the provision of eAccessibility has emerged, leading to the concepts of Universal Access and Design for All.

Foundations

Universal Access implies the accessibility and usability of Information Society Technologies by anyone, anywhere, anytime, with the aim to enable equitable access and active participation of potentially all citizens in existing and emerging computer-mediated human

activities. This can be achieved by developing universally accessible and usable products and services, which are capable of accommodating individual user requirements in different contexts of use and independently of location, target machine, or run-time environment. The discipline of Human-Computer Interaction (HCI) plays a critical role towards ensuring Universal Access to computer-based products and services, as users experience new technologies through contact with their user interfaces.

In the context of Universal Access, eAccessibility refers to the extent to which the use of an application or service is affected by the user's particular functional limitations or abilities (permanent or temporary), as well as by other contextual factors (e.g., characteristics of the environment). This implies that, for each user task of an application or service, and taking into account specific functional limitations and abilities, as well as other relevant contextual factors, there is a sequence of input actions and associated feedback, via accessible input/output devices, which leads to successful task accomplishment.

Universal Access is predominantly an issue of design. Thus, the question arises of how it is possible to design systems that permit systematic and cost-effective approaches to accommodating the requirements of potentially all users. To this effect, the concept of Design for All has been revisited in recent years in the context of HCI [14].

Design for All, or Universal Design, is well known in several engineering disciplines, such as, for example, civil engineering and architecture, with many applications in interior design, building and road construction. In the context of Universal Access, Design for All either subsumes, or is a synonym of, terms such as accessible design, inclusive design, barrier-free design, universal design, etc., each highlighting different aspects of the concept. It has a broad and multidisciplinary connotation, abstracting over different perspectives, such as:

1. Design of interactive products, services and applications, which are suitable for most of the potential users without any modifications. Related efforts mainly aim to formulate eAccessibility guidelines and standards. This is pursued in the context of international collaborative initiatives. A significant example is the effort carried out by the W3C-WAI Initiative in the area of Web accessibility guidelines [16]. Another source of web accessibility guidance is Section 508 of the US Rehabilitation Act [15].
2. Design of products which have standardized interfaces, capable of being accessed by specialized user interaction devices. An example of this approach is the Universal Remote Console, defined as a combination of hardware and software that allows a user to control and view displays of any (compatible) electronic and information technology device or service (or "target") in a way that is accessible and convenient to the user [18].
3. Design of products which are easily adaptable to different users (e.g., by incorporating adaptable or customizable user interfaces). The latter approach fosters a conscious and systematic effort to proactively apply principles and methods, and employ appropriate tools, in order to develop interactive products and services which are accessible and usable by all citizens in the Information Society, thus avoiding the need for a posteriori adaptations, or specialized design. This entails an effort to build access features into a product starting from its conception, throughout the entire development life-cycle [12].

Independently from the approach through which it may be achieved, accessibility aims to make the user experience of people with diverse functional or contextual limitations as near as possible, in terms of task accomplishment, to that of people without such limitations. Under a Universal Access perspective, accessibility has to be "designed into" the system rather than decided upon and implemented a posteriori. This raises several requirements regarding the methods, techniques and tools which can be used to integrate accessibility throughout the development lifecycle of interactive products and services. In this context, eAccessibility implies the concurrent (or adaptation driven) availability of alternative modalities, external devices and interaction styles to accommodate different needs. Therefore, multimodality plays a very important role [3].

Key Applications

eAccessibility is relevant to all interactive applications and services addressing citizens in the Information Society, in a variety of domains of everyday life, including healthcare, access to information, education, entertainment, and public administration. Some significant examples are reported below.

Web Accessibility

Access of disabled users to the World Wide Web involves the accessibility of both web content and browsing applications (web browsers). Web content accessibility is usually addressed through conformance to guidelines [16]. Various tools are available for the automatic or semiautomatic accessibility assessment and repair of web content [17]. More recent approaches go in the direction of automatic web content transformation to more accessible versions (e.g., [8]), as well as server side automatic adaptation [1]. Accessibility of web browsers is addressed through dedicated design for particular target user groups (e.g., blind users, [10]) as well as adaptation based approaches. The universally accessible AVANTI web browser [1] provides an interface to web-based information systems for a range of user categories, including: (i) "able-bodied" people, (ii) blind people, (iii) motor-impaired people with different degrees of difficulty in employing traditional input devices.

Media Accessibility

The term "rich media" indicates a broad range of digital interactive media downloadable or embedded in a web site, including video, animations, images, and sound. This type of content can be viewed or used offline with media players. Media accessibility is usually addressed through the provision of equivalent information perceivable through different senses, e.g., captioning, audio description, subtitling, and sign language translation. This requires the availability of authoring tools which support the appropriate provision of such information (e.g., [11]). Access to rich media by people with disabilities also requires the availability of accessible media players (e.g., [10]).

Accessibility in Education

eLearning systems typically rely on repositories of online materials that are made available to learners and teachers. The accessibility of eLearning content implies more than web or media accessibility, as it requires not only equivalence of information among different modalities, but also its appropriateness for the learning experience of users with diverse abilities. Meta-data classifications are under elaboration, which allow the classification of learning content according to the needs and preferences for alternative presentations of resources, methods of controlling resources, equivalent to the resources themselves and enhancements or support required by the user [9]. Access to educational material by people with disabilities also implies accessible software for the interactive delivery of such material, as well as authoring tools supporting the provision of educational content in an accessible form (e.g., [7]).

Game Accessibility

Computer games are usually quite demanding in terms of motor, sensor and mental skills needed for interaction control, and they often require mastering inflexible, quite complicated, input devices and techniques. These facts often render games inaccessible to a large percentage of people with disabilities. Furthermore, with respect to HCI, computer games have fundamental differences from all the other types of software applications for which accessibility guidelines and solutions are becoming available. Current approaches to game accessibility include the development of mainstream games compatible with the use of assistive technologies, the development of special-purpose games, optimally designed for people with disabilities, like audio-based games for blind people and switch-based games for the motor-impaired, as well as the development of universally accessible games. Examples of the latter approach are the UA-Chess web-based chess game [5] and Access Invaders ([6] – a universally accessible version of the popular classic "Space Invaders" action game), which can be played concurrently by people with different abilities and preferences, including people with disabilities (e.g., low-vision, blind and hand-motor impaired), and Access Invaders [17], a universally accessible version of the popular classic "Space Invaders" action game.

Future Directions

In the years ahead, as a result of the increasing demand for ubiquitous and continuous access to information and services, Information Society Technologies are anticipated to evolve towards a new computing paradigm referred to as Ambient Intelligence. Such an environment will be characterised by invisible (i.e., embedded) computational power in everyday appliances and other surrounding physical objects, and populated by intelligent mobile and wearable devices. Ambient Intelligence will have profound consequences on the type, content and functionality of the emerging products and services, as well as on the way people will interact with them, bringing about multiple new

requirements for the development of Information Society Technologies. In such a dynamically evolving technological environment, accessibility and usability of such a complex technological environment by users with different characteristics and requirements can not be addressed through solutions introduced once the main building components of the new environment are in place. In such a context, the concepts of Universal Access and Design for All acquire critical importance towards streamlining accessibility into the new technological environment through generic solutions [2]. However, in the context of Ambient Intelligence, Universal Access will need to evolve in order to address a series of new challenges posed by the evolving technological environment. Such challenges include the distribution of interaction in the physical environment, the optimal degree of automation vs. human control, the identification of concrete human needs and requirements in such an environment, as well as issues related to health and safety, privacy and security, and social implications. In order to support the development of Universal Access solutions for Ambient Intelligence, new methodologies to capture requirements, appropriate development methods and tools, as well as design knowledge in the form of guidelines and standards will have to be provided, thus dramatically altering the current notion and practices of eAccessibility.

Cross-references

- ▶ Human-Computer Interaction
- ▶ Multimodal Interfaces
- ▶ Usability
- ▶ Visual Interfaces

Recommended Reading

1. Doulgeraki C., Partarakis N., Mourouzis A., Antona M., and Stephanidis C. Towards Unified Web-based User Interfaces. Technical Report 394, ICS-FORTH, Heraklion, Crete, Greece, 2007, p. 283. Available online at: http://www.ics.forth.gr/ftp/tech-reports/2007/2007.TR394_Towards_Unified_Web-based_UI.pdf.
2. Emiliani P.-L. and Stephanidis C. Universal access to ambient intelligence environments: opportunities and challenges for people with disabilities. IBM Syst. J. (Special Issue on Accessibility), 44(3):605–619, 2003.
3. Furner S., Schneider-Hufschmidt M., Groh L., Perrin P., and Hine N. Human factors guidelines for multimodal interaction, communication and navigation. In Proc. 19th Int. Symp. on Human Factors in Telecommunication, 2003.
4. Garay-Vitoria N. and Abascal J. Text prediction systems: a survey. Universal Access Inf. Soc., 4(3):188–203, 2006.
5. Grammenos D., Savidis A., and Stephanidis C. *I*UA-Chess: A universally accessible board game. In Universal Access in HCI: Exploring New Interaction Environments - Proc. 11th Int. Conf. on Human-Computer Interaction (HCI International 2005), vol. 7, C. Stephanidis (ed.). Lawrence Erlbaum, Mahwah, NJ, 2005.
6. Grammenos D., Savidis A., Georgalis Y., and Stephanidis C. *I*Access invaders: Developing a universally accessible action game. In Computers Helping People with Special Needs, Proc. Tenth Int. Conf. Springer, 2006, pp. 388–395.
7. Grammenos D., Savidis A., Georgalis Y., Bourdenas T., and Stephanidis C. Dual educational electronic textbooks: the starlight platform. In Proc. 9th Int. ACM SIGACCESS Conf. on Computers and Accessibility, 2007, pp. 107–114.
8. Hanson V.L. and Richards J.T. A web accessibility service: An update and findings. In Proc. 6th Int. ACM Conf. on Assistive Technologies, 2004, pp. 169–176.
9. IMS Global Learning Consortium, http://www.imsglobal.org/accessibility/index.html.
10. Miyashita H., Sato D., Takagi H., and Asakawa C. Making multimedia content accessible for screen reader users. In Proc. Int. Cross-Disciplinary Conf. on Web Accessibility, 2007, pp. 126–127.
11. National Center for Accesible Media, http://ncam.wgbh.org/webaccess/magpie/.
12. Stephanidis, C. (ed.). User Interfaces for All - Concepts, Methods, and Tools. Lawrence Erlbaum, Mahwah, NJ, 2001.
13. Stephanidis C. and Emiliani P.L. Connecting to the information society: a European perspective. Technol. Disabil. J., 10(1):21–44, 1999.
14. Stephanidis C., Salvendy G., Akoumianakis D., Bevan N., Brewer J., Emiliani P.L., Galetsas A., Haataja S., Iakovidis I., Jacko J., Jenkins P., Karshmer A., Korn P., Marcus A., Murphy H., Stary C., Vanderheiden G., Weber G., and Ziegler J. Toward an information society for all: an international R&D Agenda. Int. J. Human-Comput. Interaction, 10(2):107–134, 1998.
15. The Rehabilitation Act Amendments (Section 508). Available at http://www.section508.gov/.
16. W3C – WAI. (1999). Web Content Accessibility Guidelines 1.0. Available at http://www.w3.org/TR/WCAG10/.
17. Web Accessibility Evaluation Tools: Overview. http://www.w3.org/WAI/ER/tools/.
18. Zimmermann G., Vanderheiden G., and Gilman A. Universal remote console - prototyping for the alternate interface access standard. In Universal Access: Theoretical Perspectives, Practice and Experience - Proc. Seventh ERCIM UI4ALL Workshop, 2002, pp. 524–531.

EAI

- ▶ Enterprise Application Integration

EC Transactions

► e-Commerce transactions

ECA Rule Action

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition
An ECA rule action is typically arbitrary code invoked if the condition of a triggered rule evaluates to true.

Key Points
The ECA rule actions are executed in response to events triggering rules whose conditions evaluates to true. The action is executed as part of a transaction or as a transaction depending on the coupling mode in the system. One major problem is that several actions can be executed concurrently and these may be in conflict. Another major problem is that executing an action may results in events triggering rules, that is, cascading rule triggering.

Cross-references
► ECA Rules
► ECA Rule Condition

ECA Rule Condition

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition
An ECA rule condition is either a database query, a logical expression or a call to a subprogram (function or method) executing arbitrary code returning true or false. If database queries are employed, then a non-empty set is equivalent to true and an empty set is equivalent to false.

Key Points
A key issue of an ECA rule condition is that it ought to take parameters carried by the event triggering the rule that, in turn, evaluates the condition. The condition either returns a set (as a result of an SQL query) or a boolean value (as a result of evaluation of the logical expression or the execution of the subprogram). An empty set is equal to false, and a non-empty set is equal to true. Another key issue is that results of the condition evaluation can be used to optimize rule action execution in many cases. Thus, there is a need to pass parameters from the condition evaluation to the rule action execution.

Cross-references
► ECA Rules
► ECA Rule Action

ECA Rules

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Synonyms
Triggers; Event-condition-action rules; Reactive rules

Definition
An ECA rule has three parts: an event, a condition, and an action. The semantics of an ECA rule are: when the event has been detected, evaluate the condition, and if the condition is satisfied, execute the action.

Historical Background
ECA rules are used within active databases for supporting reactive behavior and were first proposed in the HiPAC project [2].

Foundations
The semantics of an ECA rule is straightforward: when an event is detected, evaluate the condition, and if the condition is true, then execute the action. There are a number of reasons why the reactive behavior is abstracted to three different parts [1]:

First of all, events, conditions, and actions have different roles. An event specifies when to trigger a rule, a condition specifies what to check, and an action

specifies what to execute in response to the event. Thus, the semantics of an ECA rule is clean, and avoids ad hoc mixing of events, conditions, and actions.

Second, the separation into events, conditions, and actions is also necessary if the application requires flexible execution semantics, i.e., coupling modes.

Third, the distinct roles facilitates optimization. The event part is a prerequisite for evaluating the condition, thus conditions are evaluated when the event occurs, and not always. If the active database only allows specification of rule conditions as query statements against the database, then the condition part is a pure question on the database state and the active database can thereby utilize well known query optimization techniques.

Finally, conceptual modeling of reactive applications is made simpler if there is a corresponding notion of events both within the application domain and within the active database.

Cross-references

► Active Database (aDB)

► Active Database (Management) System (aDBS/aDBMS)

Recommended Reading

1. Dayal U. Ten years of activity in active database systems: what have we accomplished? In Proc. First Int. Workshop on Active and Real-Time Database Systems, 1995, pp. 3–22.
2. Dayal U., Blaustein B., Buchmann A., et al. S.C. HiPAC: a research project in active, time-constrained database management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology. Cambridge, MA, USA, 1988.

ECM

► Enterprise Content Management

e-Commerce Transactions

Jari Veijalainen
University of Jyvaskyla, Jyvaskyla, Finland

Synonyms

Electronic commerce transactions; EC transactions

Definition

"An electronic transaction is the sale or purchase of goods or services, whether between businesses, households, individuals, governments, and other public or private organizations, conducted over computer mediated networks. The goods and services are ordered over those networks, but the payment and the ultimate delivery of the good or service may be conducted on or off-line." [1]

A mobile e-Commerce transaction is an e-Commerce transaction that is initiated and performed using a *mobile device*, such as a mobile phone or a laptop, over a *wireless access network* or a *short-range wireless link*.

Key Points

E-commerce transaction was defined by OECD [1] from business and statistical perspective. The parties above can be individual customers (C), companies (B), or governments (A). One speaks accordingly about B2C, B2A, B2B, C2C e-Commerce and e-Commerce transactions. Historically, Electronic Data Interchange (EDI) was the first technology used in B2B and B2A e-commerce since 1970's. EDI is based on a number of agreed-upon (EDIFACT) message types whose instances are exchanged by the parties. Notice that B2C, B2A, B2B and C2C e-Commerce are usually governed by different laws. HTTP, HTML, and WWW server and browser technology matured during 1990's and E-commerce for individual customers (B2C, C2C) became possible. Physical (tangible) goods need to be delivered through a separate logistics channel, whereas digital information ("intangible goods") can be delivered through the same digital network as the order. Especially the global digital network ("Internet"), simultaneous emergence of suitable digital contents (software, music, videos, maps, etc.), and global payment infrastructure (credit cards, international banking) have made global digital B2C and other e-Commerce to explode since mid 1990's. Perceived as a distributed application one can require that a typical (B2C) e-Commerce transaction, during which a customer buys a good or goods, must satisfy *money atomicity, goods atomicity, and certified delivery* [2]. A more technical view is that an e-Commerce transaction is in general implemented as a distributed, hierarchical *workflow* crossing organizational borders. While performed, it must satisfy semantically defined atomicity constraints at each level [3]. Money atomicity and other desired properties follow from the constraints

and thus the system implementation must enforce them. The challenges are that the steps must be performed by autonomous organizations, the workflow can last days or weeks (order book from USA to Europe) and that the originating entity (customer's terminal) cannot directly control the emerging execution or its duration, because it cannot decide or even know the emerging structure of the workflow. Rather, the latter is dynamically decided by the seller and its subcontractors (suppliers). A further challenge is that the customer can later optionally cancel the order and return the received goods to the seller. The time window typically varies from 7 to 30 days depending on the local consumer protection legislation. Conceptually, this possible phase belongs to the E-commerce transaction, but in practice this is taken care of by a distinct exception handling transaction that is often semi-automatically or manually performed.

Cross-references

- ▶ Atomicity
- ▶ Workflow Schema
- ▶ Workflow Transactions

Recommended Reading

1. Annex 4. The OECD definitions of Internet and E-commerce transactions. Available at: http://www.oecd.org/dataoecd/34/16/2771174.pdf
2. Tygar D. Atomicity in electronic commerce, Chapter 33. In Internet Besieged, P. Denning, D. Denning (eds.). ACM Press and Addison Wesley, USA, 1997, pp. 389– 405.
3. Veijalainen J., Terziyan V., and Tirri H. Transaction management for M-commerce at a mobile terminal. Electronic Commerce Research and Applications., 5(3):229–245, 2006.

Eddies

▶ Adaptive Query Processing

Edge Detection

▶ Image Segmentation

eDictionary

▶ Electronic Dictionary

eEncyclopedia

▶ Electronic Encyclopedia

EERM, HERM

▶ Extended Entity-Relationship Model

Effectiveness Involving Multiple Queries

Eric C. Jensen[1], Steven M. Beitzel[2], Ophir Frieder[3]
[1]Twitter, Inc., San Francisco, CA, USA
[2]Telcordia Technologies, Piscataway, NJ, USA
[3]Georgetown University, Washington, DC, USA

Synonyms

Relevance evaluation of IR systems

Definition

In information retrieval (IR), effectiveness is defined as the relevance of retrieved information to a given query. System effectiveness evaluation typically focuses on the problem of document retrieval: retrieving a ranked list of documents for each input query. Effectiveness is then measured with respect to an environment of interest consisting of the populations of documents, queries, and relevance judgments defining which of these documents are relevant to which queries. Sampling methodologies are employed for each of these populations to estimate a relevance metric, typically a function of precision (the ratio of relevant documents retrieved to the total number of documents retrieved) and recall (the ratio of relevant documents retrieved to the total number of relevant documents for that query). Conclusions about which systems outperform others are drawn from common experimental design, typically focusing on a random sample of queries, each with a corresponding value of the relevance metric. These individual measurements for each query must be aggregated across a sufficiently large sample of queries to draw conclusions with confidence.

Historical Background

Effectiveness evaluation can be divided into user-centered evaluation, where users are given access to

an information retrieval system and observed while attempting a task, and system (ad hoc) evaluation where the retrieval process is evaluated over multiple queries without user interaction. The Cranfield experiments defined information retrieval experimentation in terms of a document collection, a set of queries, and a corresponding set of relevance judgments for each of those queries [3]. However, they assumed that the relevance of every document in the collection would be judged for each query. The Text REtrieval Conference (TREC) extended this methodology to construct reusable test collections without judging every document in the collection. They achieve this by holding constant the test collection and query set, pooling the top X results (typically 100) from each system and manually judging each document in the pool as relevant or irrelevant. It has been shown that if X (the judgment depth) is sufficiently large, these collections are reusable, in that the relative effectiveness of runs from new systems over the same documents and queries can be evaluated simply by applying the existing judgments and assuming documents that are not judged are irrelevant [1]. Recent work examines how to scale evaluation to collections the size of the World Wide Web [2].

Foundations

Evaluation paradigms based on Cranfield (including TREC) make several common assumptions:

- Documents are either relevant or irrelevant (relevance is binary).
- Relevance of a given document is independent of any others (relevance is independent).
- The information need for which the query was derived does not change (static users).

Although these assumptions are obviously untrue in practice, they continue to be the basis for simplifying the evaluation problem. Various tracks at TREC have investigated search tasks where these assumptions are particularly broken. For example, navigational search, where users are looking for a particular destination as opposed to all of the information about a given topic was studied for several years [4].

The majority of published work in information retrieval leverages the TREC methodology and its reusable test collections. This is due not only to the scientific virtue of repeatable experiments, but also to the extraordinary effort required to evaluate the effectiveness of information retrieval systems. Judging more and more documents as new retrieval algorithms are evaluated is often prohibitive. However, the growth of very large document collections such as the Web has called into question the reusability of test collections developed in this manner. Rather, some results indicate that simply judging each of the top ranked results to a shallow depth for a larger number of queries may be more efficient for achieving reproducibility on such large collections [6]. Most recently, methods of incorporating automatic evaluations (such as click through evidence or known results from taxonomies) with manual ones have been developed to reach reproducible conclusions with less manual effort [5].

There are many specific metrics available for combined measurement of precision and recall, and for aggregating these measurements across queries. The F-measure is often used to combine point-wise precision and recall in tasks where only a single measurement of each is available. Evaluating ranked lists, however, provides an inherent threshold mechanism for taking many such measurements, each at a different depth into that list. Average precision combines these measurements at varying depths, and is aggregated across multiple queries by either the arithmetic mean (MAP – Mean Average Precision) or geometric one (GMAP – Geometric Mean Average Precision). Let AP_n represents the Average Precision value for a query from a set of n queries, then MAP and GMAP can be computed as follows:

$$MAP = \frac{1}{n}\sum_{n} AP_n$$

$$GMAP = \sqrt[n]{\prod_{n} AP_n}$$

R-precision takes the precision and recall measurements at depth R where R is the number of relevant documents for that query, which averages well across queries with varying numbers of relevant documents. The arithmetic mean of the R-precision values for an information retrieval system over a set of n queries is called the Average R-precision.

Other information retrieval tasks, such as navigational searches focusing on one correct answer, do not involve a recall component. These are often evaluated by the reciprocal rank of the correct result in the list. Specifically, the Reciprocal Rank value of a query is the reciprocal of the rank at which the first

relevant result was retrieved for the query. When the Reciprocal Rank values are averaged across multiple queries, the measure is called the Mean Reciprocal Rank (MRR).

Key Applications

As in any field, improvements in information retrieval depend on the ability to evaluate. Researchers continue to leverage these techniques to refine retrieval algorithms. Search is also one of the most used features on the web, making evaluating its effectiveness critical.

Data Sets

Text REtrieval Conference test collections: http://trec.nist.gov.

URL to Code

TREC evaluation metric calculator: http://trec.nist.gov/trec_eval/.

Cross-references

- ▶ Average Precision
- ▶ Average Precision Histogram
- ▶ Average R-Precision
- ▶ GMAP (Geometric Mean Average Precision)
- ▶ MAP (Mean Average Precision)
- ▶ MRR (Mean Reciprocal Rank)
- ▶ Precision-Oriented Effectiveness Measures
- ▶ R-Precision
- ▶ Standard Effectiveness Measures

Recommended Reading

1. Buckley C. and Voorhees E.M. Evaluating evaluation measure stability. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 33–40.
2. Clarke C.L.A., Craswell N., and Soboroff I. Overview of the TREC 2004 Terabyte track. In Proc. 15th Text Retrieval Conference, 2006, NIST Special Publication.
3. Cleverdon C.W., Mills J., and Keen E.M. Factors determining the performance of indexing systems. Cranfield CERES: Aslib Cranfield Research Project, College of Aeronautics, Cranfield. vol. 1: Design, vol. 2: Results, 1996.
4. Hawking D. and Craswell N. Overview of the TREC 2001 web track. In Proc. 10th Text Retrieval Conference, 2001. NIST Special Publication.
5. Jensen E.C., Beitzel S.M., Chowdhury A., and Frieder O. On repeatable evaluation of search services in dynamic environments. ACM Trans. Inf. Syst., 26(1), 2007.
6. Sanderson M. and Zobel J. Information retrieval system evaluation: effort, sensitivity, and reliability. In Proc. 28th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 162–169.

EF-Games

▶ Ehrenfeucht-Fraïssé Games

egd

▶ Equality-Generating Dependencies

EHR

▶ Electronic Health Record

Ehrenfeucht-Fraïssé Games

Nicole Schweikardt
Johann Wolfgang Goethe-University Frankfurt am Main, Frankfurt, Germany

Synonyms

Ehrenfeucht games; EF-games

Definition

The Ehrenfeucht-Fraïssé game (EF-game, for short) is played by two players, usually called the *spoiler* and the *duplicator* (in the literature, the two players are sometimes also called *Samson* and *Delilah* or, simply, *player I* and *player II*). The board of the game consists of two structures $\mathcal{A}$ and $\mathcal{B}$ of the same vocabulary. The spoiler's intention is to show a difference between the two structures, while the duplicator tries to make them look alike.

The rules of the classical EF-game are as follows: The players play a certain number r of rounds. Each round i consists of two steps. First, the spoiler chooses either an element a_i in the universe of $\mathcal{A}$ or an element b_i in the universe of $\mathcal{B}$. Afterwards, the duplicator chooses an element in the other structure, i.e., she chooses an element b_i in the universe of $\mathcal{B}$ if the spoiler's move was in $\mathcal{A}$, respectively, an element a_i in the universe of $\mathcal{A}$ if the spoiler's move was in $\mathcal{B}$.

After r rounds, the game finishes with elements $a_1,...,a_r$ chosen in $\mathcal{A}$ and $b_1,...,b_r$ chosen in $\mathcal{B}$, and exactly one of the two players has won the game. Roughly speaking, the duplicator has won if and only

if the structures $\mathcal{A}$ and $\mathcal{B}$, restricted to the elements chosen during the rounds of the game, are indistinguishable. To give a precise description of the winning condition let us assume, for simplicity, that the vocabulary of the structures $\mathcal{A}$ and $\mathcal{B}$ only contains relation symbols. Precisely, the duplicator has won the game if and only if the following two conditions are met: (i) for all $i,j \in \{1,...,r\}$, $a_i = a_j$ iff $b_i = b_j$, and (ii) for each arity k, each relation symbol R of arity k in the vocabulary, and all $i_1,...,i_k \in \{1,...,r\}$, the tuple $(a_{i_1},...,a_{i_k})$ belongs to the interpretation of R in the structure $\mathcal{A}$ if and only if the tuple $(b_{i_1},...,b_{i_k})$ belongs to the interpretation of R in the structure $\mathcal{B}$.

Since the game is finite, one of the two players must have a *winning strategy*, i.e., he or she can always win the game, no matter how the other player plays.

Key Points

EF-games are a tool for proving expressivity bounds for query languages. They were introduced by Ehrenfeucht [1] and Fraïssé [3]. The fundamental use of the game comes from the fact that it characterizes first-order logic as follows: The duplicator has a winning strategy in the r-round EF-game on two structures $\mathcal{A}$ and $\mathcal{B}$ of the same vocabulary if, and only if, $\mathcal{A}$ and $\mathcal{B}$ satisfy the same first-order sentences of quantifier rank at most r (recall that the quantifier rank of a first-order formula is the maximum nesting depth of quantifiers occurring in the formula). This is known as the *Ehrenfeucht-Fraïssé Theorem*, and it gives rise to the following methodology for proving inexpressibility results, i.e., for proving that certain Boolean queries cannot be expressed in first-oder logic: To show that a Boolean query Q is *not* definable in first-order logic, it suffices to find, for each positive integer r, two structures $\mathcal{A}_r$ and $\mathcal{B}_r$ such that (i) $\mathcal{A}_r$ satisfies query Q, (ii) $\mathcal{B}_r$ does not satisfy query Q, and (iii) the duplicator has a winning strategy in the r-round EF-game on $\mathcal{A}_r$ and $\mathcal{B}_r$.

Using this methodology, one can prove, for example, that none of the following queries is definable in first-order logic: "Does the given structure's universe have even cardinality?", "Is the given graph connected?", "Is the given graph a tree?" (cf., e.g., the textbook [4]).

In fact, the described methodology is the major tool available for proving inexpressibility results when restricting attention to *finite* structures. Applying it, however, requires finding a winning strategy for the duplicator in the EF-game, and this often is a non-trivial task that involves complicated combinatorial arguments. Fortunately, techniques are known that simplify this task, among them a number of sufficient conditions (e.g., *Hanf-locality* and *Gaifman-locality*) that guarantee the existence of a winning strategy for the duplicator (see e.g., the survey [2] and the textbook [4]).

Variants of EF-games exist also for other logics than first-order logic, e.g., for finite variable logics and for monadic second-order logic (details can be found in the textbook [4]).

Cross-references

- ▶ Expressive Power of Query Languages
- ▶ FOL formulae syntax
- ▶ Locality
- ▶ Logical Structure

Recommended Reading

1. Ehrenfeucht A. An application of games to the completeness problem for formalized theories. Fundamenta Mathematicae, 49:129–141, 1961.
2. Fagin R. Easier ways to win logical games. In Descriptive Complexity and Finite Models, N. Immerman, P.G. Kolaitis (eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science 31. American Mathematical Society, 1997, pp. 1–32.
3. Fraïssé R. Sur quelques classifications des systèmes de relations. Université d'Alger, Publications Scientifiques, SérieA(1):35–182, 1954.
4. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.

Ehrenfeucht Games

- ▶ Ehrenfeucht-Fraïssé Games

Electronic Commerce Transactions

- ▶ e-Commerce transactions

Electronic Data Capture

- ▶ Clinical Data Acquisition, Storage and Management

Electronic Dictionary

Robert A. Amsler
CSC, Falls Church, VA, USA

Synonyms

eDictionary; Machine-readable dictionary (MRD); Terminological database

Definition

An electronic dictionary contains lexicographic information that is stored and accessed via a computer. The term "electronic dictionary" may refer to the data alone (e.g., a machine-readable dictionary), but more typically refers to a software or software/hardware system that provides access to dictionary data. Although a dictionary may include encyclopedic entries, it is typically distinct from an electronic encyclopedia. Terminological databases are special-purpose dictionaries that are used primarily to distinguish domain-specific terminology and choose appropriate terms when translating technical documents. Thesauri are special-purpose word books organized by relationships between words. They may contain definitions, but most famously, Roget's Thesaurus (available in electronic form from Project Gutenberg) does not.

Historical Background

The first widely available electronic data from print dictionaries were produced in 1966–1968 by John Olney at System Development Corporation. At that time, compositor tapes were created by commercial printers solely for their use in printing books under contract with publishers. Printers encoded typesetting information in proprietary formats. Even dictionary publishers had no access to the electronic form of the data. Therefore, to create an early electronic dictionary, Olney supervised the transcription of the contents of the Merriam-Webster New Pocket Dictionary (MPD) and the Merriam-Webster Seventh Collegiate Dictionary (W7) into punch-card images, which he subsequently distributed under license as multi-reel magnetic tape data sets.

As publishers began to realize the value of retaining the content of their dictionaries in electronic form, they began requesting printers to provide a digital copy of the typesetting data along with the print copies of the dictionary. The version of the Collins English Dictionary distributed by the Oxford Text Archive illustrates non-standard print formatting codes as used by commercial print houses.

Subsequently, publishers embraced text encoding standards, such as SGML. They teamed up with computer scientists to convert many of their books, including dictionaries, to use such standard formatting, and requested printers to typeset their dictionaries from these standard formats. The largest project to convert a legacy print text dictionary into contemporary markup text data was carried out by the Oxford University Press, in conjunction with IBM and the University of Waterloo, to computerize the Oxford English Dictionary [7].

The advent of the personal computer and CD-ROM readers led to the direct sale of CD-ROM versions of print dictionaries with proprietary interfaces and encodings of the text content. The World Wide Web also provided publishers with the opportunity to sell subscription-based access to electronic monolingual and bilingual dictionaries.

In 1978–1980, Amsler [2] directed a project at the Linguistics Research Center of The University of Texas at Austin to produce the first taxonomically structured dictionary from the Olney 1968 MPD. The project employed graduate students to manually "disambiguate" the kernel terms in dictionary definitions, providing the basis for connecting the dictionary entries defined by those terms into a "tangled" hierarchy. Olney had proposed such an enterprise a decade earlier, but had run out of funding before it could be undertaken.

Subsequent to Amsler's work, Roy Byrd [3] at IBM began a comprehensive program of computational lexicology work to use the contents of the W7 for computational linguistics. Using a new parsing algorithm for dictionary definitions, Byrd et al. also created a taxonomically structured version of the W7, but due to copyright restrictions neither the Amsler nor the Byrd taxonomic dictionaries were ever redistributed.

In 1986, George Miller planned the creation of the WordNet electronic dictionary to test psychological theories of semantic memory [4]. WordNet Version 1.0, released in 1991, offered alphabetic and taxonomic access to English definitions. Miller avoided copyright issues by creating his own definition texts, making it possible to widely distribute WordNet for others to use in academic and commercial ventures. WordNet has spawned the creation of other special-purpose electronic dictionaries, including monolingual and

bilingual WordNets for languages other than English. It remains the major free electronic dictionary available to the research community.

Foundations

Dictionaries are among the most complex documents, owing to the large number of distinct fields they contain. The structural elements found within electronic dictionaries are typically represented by XML tags or some other form of markup, such as described by the Text Encoding Initiative [6].

A typical entry in a conventional monolingual dictionary begins with a headword, usually divided into syllables, and optionally including a homograph number to distinguish the entry from others with the same headword. This is followed by the pronunciation(s), with high and low stress marks for its syllables, followed by any variant pronunciations. Next there are part-of-speech (POS) labels, other labels that reflect general usage (e.g., whether a word is formal or colloquial, whether its use is geographically restricted, and whether it is used primarily within one discipline), and inflected forms of the headword, if any. An etymology may follow, containing various language designations together with the word forms or affixes from which the headword is believed to have been derived. The etymology might also include dates and names of specific individuals when a headword's origin is reported to come from a specific person or event.

Definitions in a major dictionary are divided into distinct senses, sub-senses, sub-sub-senses, etc., down several levels. The order of the senses in a dictionary is typically either historical, with the earliest senses first, or prioritized from most common to least common senses. Each sense may include labels that document its use as being restricted to a particular region or to a particular discipline, and it may be illustrated by one or more example sentences. In the oldest dictionaries, examples were taken from notable authors, but this practice has given way to artificially constructed examples designed to be simplified illustrations of the meaning and usage of the sense. Definitions may also contain cross-references to specific senses of related headwords.

Following the definitions may be several additional entry features. These include so-called run-on entries, which are morphologically related extended forms of the headword that do not warrant their own entries elsewhere in the dictionary. These will typically have their own syllables, parts of speech and pronunciations listed. There may also be run-on entries for common phrases. Finally, an entry may also contain special treatments of synonyms and antonyms for the headword.

Key Applications

In addition to the use of dictionaries to look up individual words and their meanings (or translations), electronic dictionaries have been used in two major application areas. The first was to better understand the nature of lexical semantics through construction of taxonomies out of dictionary definition texts [2]. The second was to utilize electronic dictionaries for computational linguistic tasks ranging from morphological analysis and speech generation through word sense disambiguation. The former purpose led to WordNet and the growth of ontologies in artificial intelligence. The latter, while initially seen as promising, led to disappointment on the part of many computational linguists when they found print dictionaries inadequate to the tasks of reliably performing all computational linguistic processing tasks. However, the introduction of large electronic dictionaries changed the nature of computational linguistic system development to step away from "toy system" lexicons of a few hundred words to forever establish that systems had to be prepared for electronic dictionary-sized lexicons. Subsequently some computational linguists have embarked on creating specialized computer-based lexical resources [5]: electronic dictionaries that exist independently of the publishing community and contain advanced syntactic and semantic information.

Cross-references

- ▶ Document Databases
- ▶ Document Representations (Inclusive Native and Relational)
- ▶ SGML
- ▶ XML

Recommended Reading

1. Amsler R.A. Machine-readable dictionaries, Chapter 6. In Annual Review of Information Science and Technology (ARIST), M.E. Williams (ed.). Knowledge Industry Publications, vol. 19, 1984.
2. Amsler R.A. The Structure of the Merriam-Webster Pocket Dictionary. Ph.D. Dissertation, The University of Texas at Austin, Austin, TX, 1980.
3. Byrd R.J., Calzolari N., Chodorow M.S., Klavans J.L., Neff M.S., and Rizk O.A. Tools and methods for computational lexicology. Comput. Linguistics, 13(3–4):219–240, 1987.

4. Fellbaum C. WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA, 1998.
5. Sérasset G. Recent trends of electronic dictionary research and development in europe. Technical Memorandum, Electronic Dictionary Research (EDR). Tokyo, Japan, 1993, p. 93.
6. TEI Consortium. Dictionaries. TEI P5: Guidelines for Electronic Text Encoding and Interchange. Available online at: http://www.tei-c.org/release/doc/tei-p5-doc/html/DI.html (accessed on February 21, 2008).
7. Weiner E.S.C. Computerizing the Oxford English Dictionary. Scholarly Publishing 16(3):239–253, 1985.

Electronic Encyclopedia

Robert A. Amsler
CSC, Falls Church, VA, USA

Synonyms
eEncyclopedia

Definition
An electronic encyclopedia, like its print counterpart, provides extensive information covering general knowledge or specific disciplines through a vast collection of small articles, typically arranged alphabetically or indexed by title. Initially, electronic encyclopedias were restricted to ASCII encodings of the text portions of entries in print encyclopedias. Advances in the representation and presentation of universal character sets, images, sound, video, and interactive graphics allow today's electronic encyclopedias to present multimedia articles that are competitive with those found in conventional print encyclopedias.

Key Points
Electronic encyclopedias were initially computer-based delivery systems for their printed counterparts, accessed primarily via optical media or over computer networks. More recently, Wikipedia has demonstrated the effectiveness of collaborative co-authoring to create an electronic encyclopedia that has no print equivalent.

Encyclopedias need to be current: if not continually updated, they become obsolete with each change in geo-politics, scientific advance, or the death of a prominent individual. Furthermore, whereas print encyclopedias have elaborate indexes, taking up one or more volumes of their printed text, they cannot support compound searches. By supporting Boolean search, an electronic encyclopedia offers the ability to combine multiple concepts unanticipated by the encyclopedia editors. For example, it can support searches for entries that mention both (Winston) Churchill and (Albert) Einstein, and reveal that both won Nobel Prizes.

Entries in an electronic encyclopedia are typically multimedia articles, having text encoded with XML. Many articles also contain hyperlinks to other entries (or even to other potential entries that do not yet exist, as in Wikipedia).

Unlike commercial encyclopedias, Wikipedia is a free Web-based encyclopedia that allows readers to create and modify its content. For example, if a Wikipedia reader follows a link to a non-existent entry, the software invites the creation of the entry and will immediately accept any text offered as the body for that entry. Readers are also encouraged to update any entry deemed to be incorrect or out of date. This form of bootstrapping provides for the rapid creation of numerous entries and the editing of information in existing entries. Each editing change becomes a new entry, but can be "reverted" back to the prior entry by a future editor. Entries also have discussion pages on which authors and potential editors can comment on their content or explain their editing changes. Modification of an entry can trigger the sending of email to prior authors if they so choose, which gives authors some control over what happens to their text. In the event there are irreconcilable differences between multiple authors, the Wikipedia core group can lock down the article text and request authors to instead debate the changes in the comments pages.

Due to its large user base and the ability to monitor and revert entries to prior states, Wikipedia has flourished with millions of entries in multiple languages. However critics have questioned how it can achieve and maintain a high level of accuracy in the absence of authoritative controls over its contributors.

As well as including conventional encyclopedic information, Wikipedia has achieved broad coverage of popular culture, much of which has traditionally been regarded as trivia and unsuitable for inclusion in a print encyclopedia. Yet, for decades there have been specialty publications on trivia, crossword puzzle terminology, and special reference works for every conceivable category of information. Thus Wikipedia is more than an alternative to traditional encyclopedias: it is a potential replacement for all reference works.

The Wikipedia software is also freely available and has spawned the growth of numerous additional Wiki-based encyclopedias on the World Wide Web. Wiki software is used behind the scenes in many informational web sites. Modifications to the permissions allow Wiki software to restrict changes in content to select groups and thus permit its use in a variety of circumstances.

Cross-references

- ▶ Document Databases
- ▶ Electronic Dictionary
- ▶ Hypertexts
- ▶ Multimedia Databases

Recommended Reading

1. Giles J. Special report: internet encyclopaedias go head to head. Nature, 438:900–901, 2005.

Electronic Health Record

Amnon Shabo (Shvo)
IBM Research Lab-Haifa, Haifa, Israel

Synonyms

Virtual health record; Shared health record; Longitudinal health record; EHR

Definition

An electronic health record (EHR) is a standard-based machine-processable information entity consisting of health data pertaining to an individual. It is a result of an exhaustive aggregation of personal health data, which is longitudinal, cross-institutional and multimodal.

Historical Background

The term "medical record" usually refers to any recording/documentation of medical care or services given to a patient. The use of computerized patient records within healthcare enterprises is already a common and well-appreciated practice that has started about four decades ago [8]. It facilitates the documentation process required for medico-legal reasons and administrative procedures [10]. It also increases the availability of clinical data at the point of care which includes medical history, lab results, diagnostic imaging and so forth. The availability of such data at the point of care could help avoiding redundant tests thus saving time and cutting costs. It could also reduce medical errors and increase the overall quality and safety of care [9].

Nevertheless, medical records are typically partial and contain incomplete and inconsistent data due to fragmentation in the healthcare arena [5] where data is created in disparate systems of various healthcare providers, from single-physician offices to expert clinics, hospitals and large HMOs (health managed organizations) to name just a few (Note that the generic term "medical record" is used to refer to records also known by the acronyms Electronic Medical Record (EMR), Computerized Patient Record (CPR), and Electronic Patient Record (EPR)). In addition, the computerization of the traditional paper-based records and charts has been slow thus far and the user interfaces of many clinical information systems have been poorly designed causing time-pressed clinicians to avoid the use of it altogether.

These factors and others made the benefit of computerized patient records quite limited. This situation gave rise to the emerging concept of an electronic health record (EHR). An ISO technical report [11] states that "Previous attempts to develop a definition for the Electronic Health Record have foundered due to the difficulty of encapsulating all of the many and varied facets of the EHR in a single comprehensive definition." The report lists several definitions of EHR which demonstrate the various aspects of EHRs versus medical records such as the focus on sharing data to support integrated care. In an attempt to consolidate the various definitions, the ISO report presents the following combined definition: "A repository of information regarding the health of a subject of care in computer processable form, stored and transmitted securely, and accessible by multiple authorized users. The EHR has a standardized information model which is independent of EHR systems. Its primary purpose is the support of continuing, efficient and quality integrated healthcare and it contains information which is retrospective, concurrent and prospective."

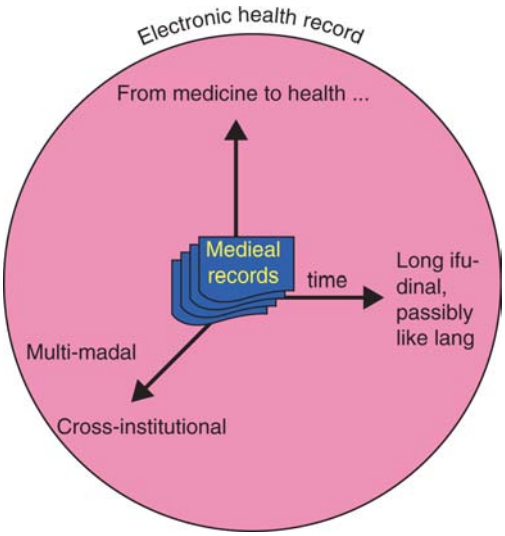
As a result of the above challenges, there have been several attempts in the past decade to realize the emerging EHR concept through national EHR efforts (e.g., in the UK [3]), regional health information

exchange efforts (e.g., in the US [1]) as well as through eHealth initiatives (e.g., the Danish national eHealth portal [2]). Most of those attempts are still facing challenges due to several reasons ranging from technical issues in integrating data from dispersed sources, to the semantic differences between various formats used by healthcare providers and to socio-economic and medico-legal issues such as privacy, ownership of the EHR, access rights, governance and business models of organizations undertaking the sustainability of patient-centric EHRs.

Foundations

The term "health record" refers to an information entity which extends the traditional medical record entity described in the previous section. The extension occurs along three dimensions (Fig. 1):

1. Content: A health record contains any health-related information including self documentation, life style, environmental data, personal preferences, etc. and is not limited to medical data as in medical records.
2. Source: While a medical record is created by a healthcare provider in order to record the care or services given to a patient by that provider, a health record is a broader information entity in the sense that it aggregates recordings created by all healthcare providers regarding that patient as well as other sources of data pertaining to the health of the patient.
3. Time: A health record is a longitudinal information entity aggregating data created possibly throughout the lifetime of the individual, while a medical record is typically a recording of medical care or services given to a patient at a specific point in time.



Electronic Health Record. Figure 1. Extending the traditional enterprise medical record information entity to the all-encompassing EHR.

It is important to note that this encyclopedia entry describes the EHR information entity and does not deal with information systems that handle medical and health records. Most of these "EHR Systems" maintain medical records of a single enterprise and cannot cope with the EHR information entity as defined here.

The following sections describe each of the extension dimensions in more detail:

Content

Extending the content of medical records to any health-related data involves the integration of data types that have not been traditionally captured in medical records such as the self documentation of habits and events as experienced by the individual, whether instructed by a healthcare professional or self initiated. Other types of data include environmental and workplace related data that can provide context to clinical data and may support reasoning about possible diagnoses and treatments. A great challenge is the personal genetic data (e.g., personal DNA variations) that can assist in the realization of the personalized medicine vision [4]. While genetic data seems yet another type of lab results, there are many differences that make it distinct and rarely captured in common medical records. In particular, the notion of genomics refer to new tests and assays applied to multiple genes interacting through complex biological pathways where personal changes in the DNA sequence as well in the expression levels of coding DNA and resulting polypeptides have significant impact on the health of that individual. The challenge in this type of data is not only its complexity and rapid changing nature but also the association to phenotypic data, whether observed in the patient or scientifically known as possible interpretations of the genetic/genomic observations. The creation of genotype-phenotype associations relevant to an individual is perhaps the greatest challenge of the emerging health record concept. Only coherent and standard presentation can enable the safe

and efficient use of this diverse data set to the benefit of the patient [13].

Source

Extending the typical single-enterprise source of medical records involves sharing data from multiple healthcare providers, expert clinics, community health services, clinical testing laboratories, diagnostic imaging facilities and any other institute that creates data about an individual. Sharing data for providing better care to a patient is challenging both semantically and technically wise.

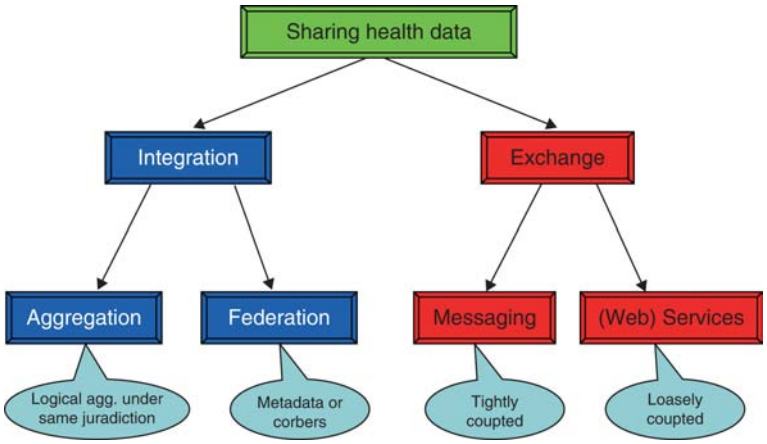
At the semantic level, the challenge is to reconcile the various formats by which the various medical records are created. The format of a medical record includes the structure (e.g., database schema, message definition, etc.) and values assigned to data items in a given structure (numeric value, medical codes, time spec, free text, etc.). To come up with a meaningful longitudinal view of the health record, it is essential to normalize the various proprietary representations of the source data; however, the lack of clear definition of the record structure could seriously hinder such effort.

At the technical level, the challenge is to find the most appropriate approach to fulfilling the requirements of information systems which handle electronic health records. The two main approaches of sharing data are exchange and integration. Data exchange is typically limited to a small number of systems exchanging information (e.g., point-to-point messaging) although the emerging service oriented architecture makes the service-based exchange more flexible.

Data integration is the preferred approach for the creation of EHRs. In principle, integration can be achieved by means of federation or aggregation (Fig. 2). In data federation (also known as "integration on the glass"), the source data is typically maintained by the enterprise where it was created and the EHR is compiled "on demand" by accessing the various sources, retrieving the data and presenting the compilation to the user. Curation and normalization are difficult to achieve at the point of care when interactive response times are expected. Efforts have been made to reach a consensus around federated formats suitable for clinical decision support and these formats are usually called Virtual Medical records (VMR). In contrast, data aggregation can be done in an on-going fashion bringing all source data into the EHR and reconciling each new item with the existing normalized record. At the point of care when the EHR is requested, an information system that aggregated the data can serve a coherent EHR in a timely fashion. The caveats of data aggregation have to do mainly with legal issues such as the release of health data by a healthcare provider and addressing privacy issues appropriately so that the EHR is highly secured and only authorized access is allowed based on proper consent of the patient. Note that the technical issues of data integartion have many governance implications and are influenced by national/regional policies as well as considerations at the socio-economic and medico-legal level [14].

Time

Medical records are typically episodic, describing visits, operations, hospitalizations, consultations and



Electronic Health Record. Figure 2. Major paradigms of data sharing through which an EHR could be created.

other services occurred in specific points in time. An EHR is a longitudinal aggregation of health data that embeds the temporal medical records along with other relevant data in a way that provides the opportunity to computationally create summaries of the data such as (i) non-redundant lists of essential data such as allergies, immunizations, diagnoses, medications etc. and (ii) useful summaries of the data such as a topical view (e.g., all data pertaining to a specific disease of the patient) or an event view (all data pertaining to a trauma the patient has gone through).

Unlike medical records, the EHR is ideally a single information entity per individual and its main value added is that it enables the creation of information layers atop of raw data (Fig. 3).

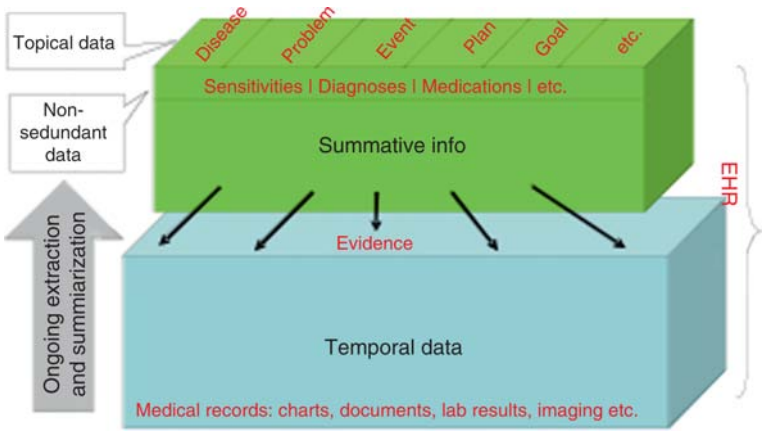
Key Applications

The main application of an electronic health record is an EHR system – a computerized information system that handles EHR information entities. EHR systems are expected to create, maintain and serve patient-centric EHRs to all authorized parties. Example of such systems are those built within the recent national and regional efforts across the globe aiming at sharing of health information in order to provide better care for the individual. Other emerging systems related to EHR are those handling the so-called Personal Health Records (PHR) – a type of EHR which is typically owned and controlled by the individual with emphases on the personal aspects in EHR such as self documentation, life style, preferences, and the like.

To overcome the diversity of formats used in the various sources integrated by an EHR system it is crucial to use standardized formats, preferably internationally-recognized EHR standards. Indeed, in the recent years, two significant standards have been developed and published by international standards development organizations. Health Level Seven coordinated an intensive international effort to publish a Functional Model Standard for EHR [7] and the European Standardization Body (CEN) has finalized an information model for EHR along with a new constraining formalism [12] to enable the standard representation of common health data sets within the EHR.

A somewhat related standard is the Clinical Document Architecture (CDA) that provides an electronic counterpart of the so common paper-based clinical documents such as operative notes, discharge summaries, referral letters and so forth [6]. These documents are a natural input to the longitudinal EHR as they typically include summary of patient's health conditions. All those standards complement each other and lay the ground for EHR Systems to finally be ready to achieve the ultimate goal of a patient-centric, standard-based, longitudinal and cross-institutional electronic health record.

A key application in the area of sharing and managing clinical documents as a step towards the EHR is an integration profile that has been developed by the IHE (Integrating the Healthcare Enterprise) and is coined XDS (Cross-enterprise Document Sharing [15]). The XDS integration profile is in fact a workflow specification that a clinical affinity domain (CAD) employs in order to share clinical documents created in disparate locations about the same patient. A CAD is established by enterprises that create and maintain



Electronic Health Record. Figure 3. Layers of temporal and summative data constituting the EHR.

clinical documents and decided to share the documents through the affinity domain. Each CAD has a centralized registry where metadata about each document (e.g., the patient, author, authenticator, encounter, services, etc.) are registered by "document sources" so that queries can be posed by "document consumers." The retrieved data from the registry includes merely the metadata and once the document consumer selects documents of interest, the registry then approaches the relevant documents repositories residing in the respective premises of the CAD cooperating enterprises. Given proper authorization of the document consumer, the selected documents are then presented to the consumer. These documents could then be further processed to provide summary or analysis content per the user request. Such processes are complex to perform "on-the-fly" but the hope is that simple analyses could show the benefits of EHR to users who then might demand the creation of a coherent patient-centric EHR out of all available clinical documents and other types of medical records.

It is important to note that the XDS architecture doesn't allow for fully querying of the clinical documents stored in the enterprise repositories. The query can be done only against the metadata that was registered by the document sources. Queries by clinical data that is typically held in the body of the documents cannot be queried using the XDS integration profile. Nevertheless, the addition of an EHR data model on the top of an XDS architecture could then make the XDS repositories fully structured according to an EHR data model such as that of the CEN 13606 standard. Systems with such capabilities will also lend themselves to queries which are not patient-specific, e.g., search for data by disease, pathogens, populations and even genetic data.

Cross-references

► Clinical Document Architecture

Recommended Reading

1. Adler-Milstein J., McAfee A.P., Bates D.W., and Ashish K.J. The state of regional health information organizations: current activities and financing. Health Aff., 27(1):w60–w69, 2007.
2. Britze T.H. The Danish National e-health portal – increasing quality of treatment and patient life. Technol. Health Care, 13(5):366–367, 2005.
3. Coiera E.W. Lessons from the NHS National Programme for IT. Med. J. Aust., 186(1):3–4, 2007.
4. Davis R.L. and Khoury M.J. The journey to personalized medicine. Personalized Med., 2(1):1–4, 2005.
5. DePhillips H.A. Initiatives and barriers to adopting health information technology: a US perspective. Dis. Manag. Health Outcomes, 15(1):1–6, 2007.
6. Dolin R.H., Alschuler L., Boyer S., Beebe C., Behlen F.M., Biron P.V., and Shabo A. HL7 clinical document architecture, release 2. J. Am. Med. Inf. Assoc., 13(1):30–39, 2006.
7. Electronic Health Record System (EHR-S). Functional requirements standard. Available at: http://www.hl7.org/EHR/. Accessed on October 28, 2008.
8. Giere W. Electronic patient information – pioneers and much-More. A vision, lessons learned, and challenges. Methods Inf. Med., 43(5):543–552, 2004.
9. Haux R., Ammenwerth E., Herzog W., and Knaup P. Health care in the information society. A prognosis for the year 2013. Int. J. Med. Inf., 66(1–3):3–21, 2002.
10. Hollerbach A., Brandner R., Bess A., Schmucker R., and Bergh B. Electronically signed documents in health care – analysis and assessment of data formats and transformation. Methods Inf. Med., 44(4):520–527, 2005.
11. ISO/TC 215 technical report. Electronic health record definition, scope, and context. Second draft, August 2003.
12. Kalra D. Electronic Health Record Standards. In: IMIA Yearbook of Medical Informatics 2006, R. Haux, C. Kulikowski (eds.). Methods Inf. Med., 45:S136–S144, 2006.
13. Shabo A. The implications of electronic health records for personalized medicine. Personalized Med., 2(3):251–258, 2005.
14. Shabo A. A global socio-economic-medico-legal model for the sustainability of longitudinal electronic health records. Methods Inf. Med., 45(3 Pt 1):240–245, Methods Inf. Med., 45 (5 Pt 2):498–505, 2006.
15. XDS - cross-enterprise document sharing. Developed by the IHE (Integrating the Healthcare Enterprise) IT Infrastructure Committees. Available at: http://www.ihe.net/IT_infra/committees/index.cfm

Electronic Ink Indexing

Walid G. Aref
Purdue University, West Lafayette, IN, USA

Synonyms

Handwritten text; Online handwriting

Definition

With the proliferation of personal digital assistants (PDAs) and pen-based handheld devices, it is important that computers understand handwritten text (or electronic ink). In this pen-based environment, one

can have handwritten file contents, handwritten file names, handwritten directory names, handwritten email messages, handwritten signatures, etc. With the large bodies of handwritten content, indexing techniques are essential in order to search for the relevant content. The fact that the data is handwritten makes the problem more difficult than in conventional situations. No two persons handwrite a word in exactly the same way. Even the same person cannot write a word in the same way twice. This makes the indexing and retrieval of handwritten data a hard problem.

Historical Background

Retrieval techniques for handwritten data have to be scalable so that they can handle the continually growing sizes of multimedia data, e.g., scanned documents in digital libraries and handwriting databases. An additional requirement for ink processing in a pen-based and/or a personal digital assistant environment is the need to support online retrieval and fast response time. There are two approaches to dealing with handwritten electronic data, namely, handwriting recognition and ink pattern matching.

Handwriting recognition is a procedure for converting pen strokes into strings of characters (or any other fixed character set). Once converted into characters, strings can be manipulated and searched in conventional ways. One problem with handwriting recognition is that it is prone to errors. Many critics point out that handwriting recognition does not meet the need of users. In fact, it is widely believed that this shortcoming is one of the main causes why the sales of pen-based tablets have fallen short of the expected figures. Moreover, even if handwriting recognition becomes highly accurate, it is clear that it provides a mapping from a highly expressive medium such as ink to a constrained medium, such as character strings. Handwriting recognition is not directly applicable to diagrams, drawings and many special symbols that the user can write.

Because of the drawbacks of handwriting recognition, the notions of *approximate ink matching* (AIM) and *pictograms* have emerged as an interesting alternative to handwriting recognition [11]. Pictograms are simply handwritten pictures, while AIM is the technique that evaluates how well two pictograms match. By using pattern matching techniques, AIM can take an input pictogram and evaluate how well it matches each one of the previously stored pictograms. AIM eliminates the problems of expressiveness, language and alphabet dependency and inaccuracy of handwriting recognition. The procedure simply focuses on finding a pictogram that resembles the input, without trying to "understand" or translate its meaning. AIM algorithms with high matching accuracy have been developed.

The problem with the AIM technique is that it could be computationally expensive. Without any other tool, AIM techniques are forced to sequentially search the entire pictogram repository. As the size of the pictogram repository grows, this process becomes painfully slow and impractical, especially that pattern matching techniques can be computationally expensive. This highlights the need for scalable techniques that provide fast response time for retrieval queries in handwritten databases of larger sizes.

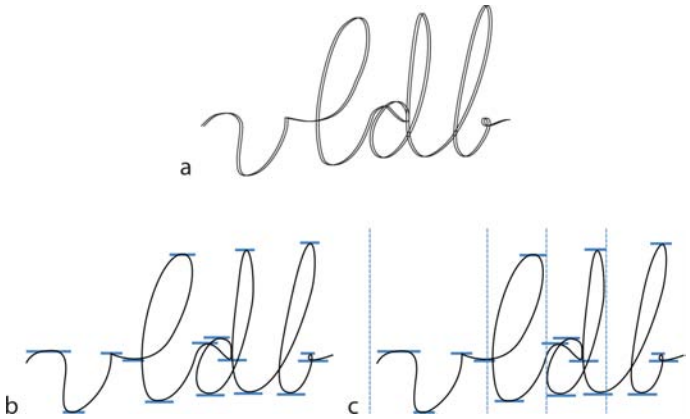
Foundations

Electronic handwritten ink in the form of a pictogram needs to be transformed into a more robust form before being passed to the database for further processing. This transformation needs to take place regardless of whether the pictrogram is to be inserted into the database, or used within a query to search the database. Electronic ink can be expressed as a sequence of time-stamped points in the plane:

$$s = (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (X_k, Y_k, t_k),$$

where (x_i,y_i) represents the location of an electronic ink pixel at time t_i, $t_i \leq t_{i} + 1$, and k is the number of pixels in the pictogram.

Given a sequence S and a database of sequences S_j $(j = 1,\ldots,M)$ an important operation is to search the database for the sequences that are similar to S. Traditionally, databases use an alphanumeric representation of the data. This representation is ideally unique, precise and stable. Electronic ink data lack these qualities, making its matching a difficult problem. The data are often corrupted with noise. Even ideal ink does not provide an adequate basis for sequence identification, because a pictogram needs to be identified given slightly different variations in its shape. Different people cannot write the same word in the same way. Even for the same person, it is very difficult to generate *exactly* the same pictogram twice (it will almost always be the case that the stroke information varies each time the person writes the same word).

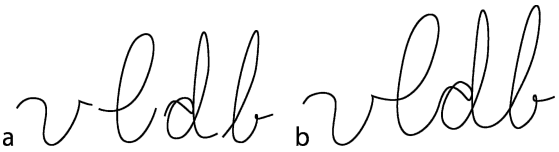


Electronic Ink Indexing. Figure 1. Example illustrating the segmentation of the pictogram in (a) into (b) strokes, and (c) alphabet symbols.

Electronic Ink Representation

In order to allow for approximate matching of handwritten electronic ink, each handwritten pictogram needs to be transformed into a more robust form. The raw electronic ink data that corresponds to points (or pixels) in the two-dimensional space is not robust as a slight shift in the input sequence may change the values of the x and y coordinates of the points representing a word. As a result, other features that are less sensitive to common deformations, e.g., translation, and rotation, are computed. Example features are the bounding box that contains all the points that form a pictogram, the average curvature, the number of points, etc.

These features can be computed at different levels of granularities. In this regard, handwritten electronic ink can be viewed at three levels of abstraction: the stroke level, the alphabet symbol level, and the pictogram level. Figure 1 illustrates a pictogram and its segmentation into pen strokes and handwritten symbols. Pictograms can be represented by any of the three granules in the figure, i.e., as one entity containing the entire pictogram (Fig. 1a), as a sequence of pen strokes (Fig. 1b), or as a sequence of alphabet symbols (Fig. 1c). For instance, in order to select the symbols as granules, a segmentation algorithm is needed to properly separate the symbols. For strokes, a simple segmentation algorithm picks local minimum (or maximum) points and uses them to segment the curve. Segmentation could be a difficult task for some types of pictograms, such as cursive handwritten words (Fig. 2b), or a simpler task as in handprinted words (see Fig. 2a). Some languages, like Japanese,



Electronic Ink Indexing. Figure 2. A handprinted word, (b) a cursive handwritten word.

lend themselves easily to symbol segmentation. In Japanese, Kanji symbols are already separated by blank spaces. The choice of granularity has an impact on the type of indexes to be built.

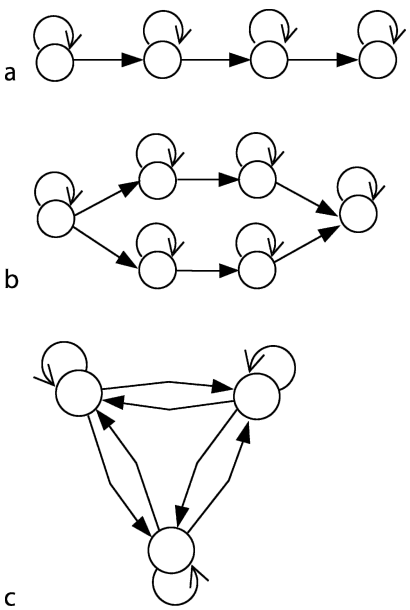
The second issue is that, for matching purposes, it is better to talk about pictogram (symbols or strokes) classes instead of individual pictograms (symbols or strokes). A pictogram class is the set of pictograms that have the same semantics, according to the user. Of course, it would be impractical to store a pictogram class by storing the list of pictograms that belong to it. Alternatively, a *representative* of the class and a *distance metric* needs to be adopted. Inputs are matched against the representative (after the necessary preprocessing) and the distance metric ranks the matches. The representative is not a pictogram, but rather a model that captures the essential qualities of the pictogram class. In the next section, a sample model is presented that can be used to represent pictograms.

Hidden Markov Models (HMM)

HMMs are already used in the field of speech and handwriting recognition as a powerful tool for

speech and handwritten document matching (see e.g., [11–13,15]). Each pictogram in the database can be modeled by an HMM. The HMM is constructed so that it accepts the specific pictogram with high probability (relative to the other pictograms in the database). In order to recognize a given input pictogram, each HMM in the database is executed and the one that generates the input sequence with the highest probability is selected. Each HMM in the underlying sequence database has to be tested, which results in a linear process, and the speed of execution becomes the primary difficulty.

An HMM is a doubly stochastic process, where there is a probability distribution that governs the transitions between states and an output probability distribution that identifies the distribution of output symbols for each state. An excellent coverage of HMMs can be found in [12,13]. For demonstration purposes, one type of HMM structures, termed *left-to-right HMM* [5] is illustrated in Fig. 3a. Left-to-right HMMs are useful for modeling temporal signals as in sound (see e.g., [13]) and cursive handwritten text (see e.g., [11]) because the underlying state sequence associated with the model has the property that, as time increases, the state index increases (or stays the same) – that is, the system states proceed from left to right.



Electronic Ink Indexing. Figure 3. (a) a left-to-right HMM model with four states, (b) a parallel path left-to-right HMM model with six states, and (c) the ergodic model with three states.

A left-to-right HMM can be constructed to model a handwritten word or an alphabet symbol. The HMM is constructed so that it accepts the word (or the symbol) with high probability (relative to the other words in the database). Training techniques may be applicable in the case where multiple instances of the word are available (e.g., a word can be handwritten multiple times, resulting in more than one sample). These samples can be used to train the HMM so that it can match similar words with higher probabilities (see e.g., [1,10]). The probability given by the HMM is the distance metric used for ranking purposes.

Given an HMM that models a word (or symbol), an input symbol can be run against the HMM to obtain as output a matching probability. Given a set of stored words (or symbols) with their associated HMMs, an input word (or symbol) can be searched by running each one of the corresponding HMMs against the input and choosing those with the best matching probability. In fact, the size of the answer set can be set as a parameter to choose the k best matches. To train HMMs and to use them to match a word or a symbol, a more robust form of ink representation needs to be used.

Representation Granularity

Features can be computed at the stroke level, pictogram level, symbol level, or through a collection of local features. For example, in the case of a stroke, a pictogram is segmented into a set of strokes, where each stroke is carefully described with a set of features. Thus, a pictogram can be represented by a collection of points in the feature space, one point per stroke. In contrast, at the pictogram level, a set of global features can be computed for the entire handwritten pictogram. Thus, each pictogram is described with a vector of feature values and is represented as one point in a multi-dimensional space.

Another simple, but efficient way of representing a pictogram is to pick some sample points from the pictogram (or symbol, or stroke) and compute some local features, ones that depend on a sample point and possibly the one (or two) surrounding points from each side. Depending on the application, some of these features may be more relevant than others, and hence not all of the features need be computed at a given point in the sequence. Common features are: direction, velocity, change in direction, change in velocity, accumulative angle (with respect to the initial

point), accumulative length and angle of bounding box diagonal (also with respect to the initial point), and accumulative sequence length [14]. As an example of computing local features, the speed f_s at point p can be computed by: $f_s = sqrt(\Delta x_p^2 + \Delta y_p^2)/\Delta t_p$, *where* $\Delta x_p = x_{p+1} - x_p$, $\Delta y_p = y_p + 1 - y_p$, and $\Delta t_p = t_p + 1$ $t_p + 1 - t_p$. After computing the local features, the pictogram can be represented by a sequence of feature vectors (v_1, t_1), (v_2, t_2),... The dimensionality of v_i corresponds to the number of local features at each point.

Indexing Techniques

Sequential search does not scale well. The process becomes unacceptably slow as the number of pictograms stored in the database grows. Therefore, indexing techniques that help prune the choices are called for. However, indexing techniques for handwritten pictograms must exhibit two characteristics that make them different from traditional indexing techniques:

1. The structures must incorporate the underlying model that is chosen to represent ink. The model must play an active part of the index.
2. Due to the high variability of the ink data, the indices must provide approximate matching.

The following sections overview two sample indexing techniques that exhibit both of the characteristics stated above.

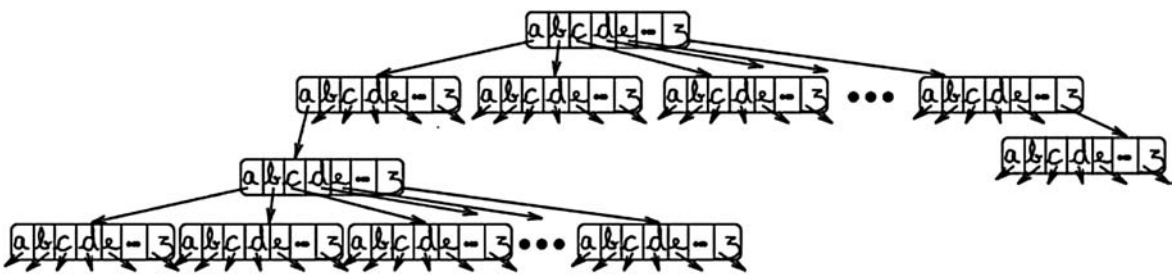
Alphabet-Level Indexing The handwritten trie [2] (Fig. 4) models ink at the alphabet symbol level and represents each of the symbol classes by using HMMs. The traditional trie data structure [4] is an M-ary tree whose nodes have M entries, and each entry corresponds to a digit or a character of the alphabet. Each node on level l of the trie represents the set of all keys that begin with a certain sequence of l characters; the node specifies an M-way branch, depending on the $(l + 1)$th character.

As pointed out in [6], the memory space of the trie structure can be significantly reduced (at the expense of running time) by using a linked list for each node, since most of the entries in the nodes tend to be empty. This results in what is termed the *forest trie*.

The handwritten trie uses a variant of the forest trie where nodes of the forest trie are packed into disk pages (a disk-based trie). Alphabet symbols of the trie are all handwritten. Since each symbol can be written differently each time, a separate HMM model per alphabet symbol is used in order to model that symbol. Given a handwritten query word, say w, a preprocessing step takes place to breakdown or segment w into a sequence of handwritten strokes or segments w_s. Various segmentation techniques can be used, e.g., [3]. Next, the handwritten trie is searched for an instance of w_s. Each of the HMMs at the root of the handwritten trie is executed against the first symbol in w_s and the search branches to the child node that has the best match. Mismatchings between w_s and items in handwritten trie are dealt with by backtracking and exploring different paths based on best matches between items of w_s and the corresponding items in the trie. An adapted version of the A* algorithm is used in order to prioritize which nodes of the trie to visit next.

In the case of the handwritten trie, matching is performed by using Hidden Markov Models (HMMs) [12]. The handwritten trie uses one type of HMMs, namely the *left-to-right HMM* [1] (see e.g., Fig. 3a), which is useful for modeling temporal signals as in sound (see e.g., [13]) and cursive handwritten text (see e.g., [11]).

The backtracking search mechanism in the handwritten trie takes care of 1-1 substitution errors, i.e.,



Electronic Ink Indexing. Figure 4. An example Handwritten Trie.

when a letter is mal-written, and hence is classified as another letter. However, there are other types of errors that are common in handwriting that needs to be dealt with. For example, if one of the symbols or letters in the query word is omitted (i.e., a *deletion* error), then an entire level of the handwritten trie is skipped and a node's grandchildren instead of its children are checked. Similarly, when an extraneous letter is introduced, e.g., due to an error in the cursive writing segmentation procedure (i.e., an *insertion* error), then the corresponding symbol of the input query word needs to be consumed without probing a node's children. To summarize, when deciding on how to match a symbol from the input query word against a node in the handwritten trie, all three types of error scenarios (1-1 substitution, deletion, and insertion errors) need to be investigated and prioritized using the A* search algorithm. Details of this procedure can be found in [4].

Stroke-Level Indexing

Indexing of handwritten text can be performed at the stroke level. A pictogram is segmented into a set of strokes. Each stroke is carefully described with a set of features, and, thus, can be stored as points in the feature space. Subsequently, any multi-dimensional access method, such as the R-tree, can be used. Similarity-based retrieval can be performed by executing a few range queries and then applying a voting algorithm to the output to select the handwritten words in the database that are most similar to the query [8]. The main advantage of this approach is that it is able to handle substring matching efficiently. Also, in contrast to the handwritten trie, stroke-level indexes are not order-dependent. For example, in scenarios where the order of handwriting a word may differ, the difference in the order of writing the strokes that compose Chinese characters does not adversely affect the search process. The reason is that regardless of the order, each stroke is mapped to a point into the multi-dimensional space and the neighborhood of each point is search for a best match, or best-k matches. Some rank aggregation needs to take place in order to merge all the returned best-k matches and return the best matched handwritten word in the database. Several techniques can be adopted to insure that the index is resilient to the kind of errors that result from the segmentation process, namely, stroke insertion/deletion and substitution errors [9].

Pictogram-Level Indexing

At the pictogram level, a set of global features can be computed for the entire handwritten pictogram. Thus, each pictogram is described with a vector of feature values and is represented as a point in a multi-dimensional space. In contrast to indexes that are based on stroke-level representations, no voting algorithms are needed as each pictogram is represented by only one point [9]. However, in order to increase the robustness and the matching rate of this approach, a multistage retrieval algorithm is adopted. At the first stage, the index is used to reduce the search space to a small set of candidate words that are similar to the handwritten query pictogram. The candidate set is then subjected to a pipeline of two sequential search algorithms that use a different set of extracted features and a distance ranking function to extract the most similar pictogram to the handwritten query pictogram.

Handwritten Ink as a Special Form of Time-Sequence Data

It is to be observed that handwritten text is one form of time-sequence data. As a result, all the indexing techniques applicable to handwritten text can also be applicable to the more general case of sequence data. For example, modeling sequence data using HMMs and similarity search and approximate matching techniques are also applicable to general sequence data and are not limited to only handwritten text. However, the converse is not true. The techniques used for indexing and similarity-based retrieval of time sequence data need to be adapted to make use of the special nature and features of electronic ink.

Cross-references

- ▶ Document Databases
- ▶ Feature Extraction for Content-Based Image Retrieval
- ▶ Feature Selection for Clustering
- ▶ High Dimensional Indexing
- ▶ Indexing and Similarity Search
- ▶ Indexing Metric Spaces
- ▶ Multimedia Data Indexing
- ▶ Multimedia Databases
- ▶ N-gram models
- ▶ Nearest Neighbor Query
- ▶ Object Recognition
- ▶ Similarity and Ranking Operations
- ▶ Text Indexing and Retrieval

- ▶ Text Indexing Techniques
- ▶ Tree-based Indexing
- ▶ Trie

Recommended Reading

1. Aref W.G., Vallabhaneni P., and Barbara D. On training hidden Markov models for recognizing handwritten text. In Proc. Int. Workshop on Frontiers of Handwriting Recognition, 1994.
2. Aref W.G., Barbará D., and Vallabhaneni P. The Handwritten Trie: Indexing Electronic Ink. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 151–162.
3. Aref W.G., Barbara D., Lopresti D.P., and Tomkins A. Ink as a first-class multimedia object. In Multimedia Database Systems: Issues and Research Directions, S. Jajodia and V.S. Subramanian (eds.). Springer, Berlin, 1995.
4. Aref W.G. and Barbará D. Supporting electronic ink databases. Inf. Syst., 24(4):303–326, 1999.
5. Bakis R. Continuous speech word recognition via centisecond acoustic states. In Proc. 91st Mts. of the Acoustical Society of America, 1976.
6. de la Briandais R. File searching using variable length keys. In Proc. Western Joint Computer Conference, 1959, pp. 295–298.
7. Fredkin E. Trie memory, Commun. ACM, 3(9):490–500, 1960.
8. Kamel I. Fast retrieval of cursive handwriting. In Proc. Int. Conf. on Information and Knowledge Management, 1996, pp. 91–98.
9. Kamel I. and Barbará D. Retrieving Electronic Ink by Content. In Proc. Int. workshop on Muldimedia Database Systems, 1996, pp. 54–61.
10. Levinson S.E., Rabiner L.R., and Sondhi M.M. An Introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. Bell Syst. Tech. J., 62(4):1035–1074, 1983.
11. Lopresti D.P. and Tomkins A. Approximate matching of hand-drawn pictograms. In Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition, 1993, pp. 102–111.
12. Rabiner. L.R. A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE, 77(2):257–285, 1989.
13. Rabiner L.R. and Juang B.H. Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs, NJ, 1993.
14. Rubine, D.H. The Automatic Recognition of Gestures, Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, December, 1991. Also Technical Report Number CMU-CS-91-202.
15. Sin B.K. and Kim J.H. A statistical approach with HMMs for on-line cursive Hangul (Korean script) recognition. In Proc. 2nd Int. Conf. on Document Analysis and Recognition, 1993, pp. 147–150.

Electronic Libraries

- ▶ Digital Libraries

Electronic Newspapers

Robert B. Allen
Drexel University, Philadelphia, PA, USA

Definition

Newspapers collect, organize, and periodically disseminate information to a community in the form of news articles and other features. They are complex information resources which generally must also be economically viable. Electronic newspapers can provide many of the same types of services as traditional newspapers but they have many additional advantages for the readers. The electronic newspaper can be updated with the latest bulletins, tailored to the reader's interest, and take advantage of multimedia. However, increasingly, news and related services are being disaggregated and provided by separate suppliers so that in some cases, the electronic newspaper can sometimes be just a personalized web portal.

Historical Background

Newspapers arose in the seventeenth century in mercantile cities in Germany and the Netherlands. From the eighteenth to the twentieth centuries, newspapers were often the primary source of citizen's knowledge of the world beyond his immediate experience. Gradual changes in news communication technologies such as the telegraph, production technologies such as printing, delivery technologies such as trucking, increased the circulation and impact of newspapers throughout the nineteenth and early twentieth centuries. Early electronic media such as radio and television provided competition but newspapers still thrived. However, at the end of the twentieth century cable television began 24 h news coverage. The 24-h news cycle means that there is a greater need for newspapers to provide background and analysis and in many cases newspapers seem to be becoming more partisan. Interactive electronic media have amplified these trends and dramatically lowered the barriers to entry for news services. Among the notable effects has been the rise of blogging and citizen journalism. Although some electronic editions are very similar to their paper editions, in many cases it is better to consider electronic newspapers as a set of relatively loosely connected services rather than monolithic entities.

These technological upheavals have led to uncertainty as to how to maintain viable business models

for newspaper publishing. As a result of digital convergence and shift to Web distribution most news content is supported by Web advertising. Web development and management also allows the multi-purposing of news content across delivery platforms. However, the Web has lowered the barriers to entry and allowed non-traditional news sources such as blogs and citizen journalism to flourish. Thus, the emerging news sources are broader but do support the depth of news gathering that has existed in the past.

While the focus of this review is on articles, electronic newspapers incorporate many other types of content such as classifieds, financial data, editorials, movie reviews, weather, sports, cartoons, and other advertising. Indeed, the distinction between news and entertainment is becoming blurred. Moreover, because of the ease of electronic publication, many sources of news are emerging such as blogs and citizen journalism. Indeed, these many be seen broadening newspapers' role in developing an informed citizenry. While dissemination of newspapers and journalism can broaden freedom of information there remains the possibility that information technology may ultimately facilitate censorship.

Foundations

An electronic newspaper is not just the presentation of a static newspaper page on a screen. Many electronic services contribute to its production, dissemination, and access and while news articles are the defining characteristic of newspapers, there are many other components which are coordinated to make up the content.

Managing Content

Metadata A comprehensive set of metadata descriptions for news articles would include date, rights, genre, and topic. An extensive set of XML-based descriptive metadata for news has been developed by the International Press and Telecommunications Council (IPTC). Beyond the general genre and topic descriptors for individual articles there are also content tags (e.g., for named entities) and structural descriptions for tabular material such as sports scores and financial data. The IPTC descriptive systems also include metadata for news images such as metadata describing the nature of the shot. For broader applications such as long term preservation and composite news objects, a multilayer metadata description framework such as the Multimedia Encoding and Transmission System (METS) can be used.

Digital Asset Management Systems Repositories extend standard Web servers by supporting content management services. The repositories used by electronic newspapers may be best characterized as digital asset management systems. Different types of digital asset management systems typically support different types of activities. Content developers such as journalists, editors, and layout designers would use a library-like service to access stored digital resources. This could provide access to stored content in a way similar to traditional photo "morgues" used by newspapers. That is they would share aspects of library services and production management services. A second major area of application for digital asset management systems in electronic newspapers could be to organize and push content to a web site. In other words, they would provide digital supply chain services.

Automated Text Processing of News Resources

To support complex activities such as composing news stories and allowing readers to interact with them, digital asset management systems need to support intermediate-level services such as summarization, search, rights management, and preservation.

Information Extraction and Text Data Mining While some news articles are now developed and disseminated with rich metadata and semantic annotation embedded, there is still a lot of untagged text. Extracting that text from unstructured news information would allow better text processing. The primary approach is based on the extraction of named-entities. There are also other extraction techniques such as template matching. Taken together, such information extraction techniques are examples of text data mining. Eventually, such text mining might extract facts in the news articles and allow them to be used in a question-answering system. In addition, it could be used to track opinions expressed on the Web in editorials, movie reviews, and blogs.

Event Detection and Tracking Of course, the main purpose of news articles is to report events so extracting, those events will be critical to any attempt to process the news articles. Topic Detection and Tracking (TDT) uses statistical approaches similar to those from

information retrieval (IR). In this approach a topic is defined as a set of related stories. TDT is composed of several tasks such as new topic identification and topic tracking across a set of articles. The main technique is statistical clustering. However, this is a challenging task and the similarity is usually based on the entire description, not just the primary event. In order to improve accuracy, recent approaches have examined events at a finer-grained level. These approaches typically employ rule-based linguistic methods. For instance, they may attempt to determine the verb frames which are employed.

Commercial News Aggregators, News Summarization, and Searching Commercial news content aggregators collect articles from various news sources. In order to present them the aggregator generally clusters them. The most effective of these techniques perform the clustering on named-entities. The articles in each cluster are then ranked by attributes of their source, estimates of their scope and impact, and popularity and then presented to users.

Although they are not yet common in commercial news aggregation systems, summaries may be automatically synthesized for the clusters. Because this is done across news articles it is a type of multi-document summarization. Like most complex text, news text is composed of high-level rhetorical units. Identifying those units turns out to be helpful for generating summaries.

Individual articles and clusters of articles which are collected by news aggregators may also be indexed and searched. In addition to the usual parameters for weighting index terms such as term frequency and document frequency, terms in news articles can also be ranked on factors such as the frequencies and authority of the articles.

Additional Services and Issues

Preservation Services News has been called the "first draft of history" and, more generally, newspapers can be a significant cultural record. For instance, local libraries generally like to keep records of their local newspapers. This is an instance of keeping the record of the community. Unlike paper, electronic copies are ephemeral. However, with the decreasing cost of storage, it is now possible to collect and save almost all local news (with the possible exception of video). Challenges remain in collecting the materials. In addition to the preservation of electronic news materials, there is also now considerable interest in processing digitized historical newspapers. However, automatically extracting and processing the OCRd images has proven difficult because of variable quality.

Rights Management As with many types of commercially produced digital media, it remains unclear what is the best business model for the producers to recoup their expenses. Many publishers believe in maintaining tight control over their resources. This is certainly an issue for publishers with respect to the news aggregators. A related contentious rights issue concerns permission to store archival copies of the new items collected. With traditional newspapers, libraries could legally collect sets of old newspapers. At least in the U.S., libraries and archives are not currently allowed to made copies of digital objects for preservation.

Managing Multimedia News Content

Of course, a lot of news is on television and radio broadcasts and, increasingly, Web-based electronic newspapers include multimedia. The PRX and PBCore metadata standards have been developed for public broadcast radio and video. In addition, there are general multimedia metadata standard such as MPEG-7 and PRISM which can be applied productively to multimedia news objects.

As with data mining of text-based news articles, there could be great value in mining multimedia news objects. There has been considerable work on information extraction and summarization from television and radio news broadcasts. However, this is very challenging because it may involve combinations of different and complex processes including speech processing (unless closed captions are available), image recognition, and video segmentation. These processes tend to be error-prone leading to inaccuracies in named-entity extraction and this, in turn, affects the quality of services such as summarization and searching which use those named entities.

Presentation and Access Services

Personalized Interaction While traditional newspapers aim to provide news which is suitable for a wide range of readers, potentially news articles could be filtered to match a profile of personal preferences. These

personalized articles could be combined into an entire personalized newspaper. Indeed, this sort of news interface could already be constructed from articles provided by Really Simple Syndication (RSS) services.

Recommendations Beyond personalization based on an individual's own profile, there are several ways news-related behaviors can be used to increase personalization. One technique is collaborative filtering in which suggestions are made from friends and coworkers. The ubiquity of email forwarding associated with electronic news papers shows the popularity of this type of sharing.

Automated techniques for generating recommendations are based on the pattern of news article accesses by individuals who have no particular connection to the reader. Such recommendations, which are also used for movies and music, are often calculated correlations between the user's choices and those of other individuals. When massive amounts of data are available, variations of singular-valued decomposition (SVD) have proven effective. Another way to simplify the data is by not using correlation but using simple counts such as those provided by "co-visitation." In any event, there are still substantial difficulties in generating effective recommendations such as the stability of people's interests ensuring unobtrusiveness and privacy ensuring in the collection of those interests, and the cold-start problem which occurs when there are no previous observations.

Cross-references

- ▶ Digital Archives and Preservation
- ▶ Digital Libraries
- ▶ Digital Rights Management
- ▶ Indexing the Web
- ▶ Information Extraction
- ▶ Information Retrieval
- ▶ Metadata
- ▶ Multimedia Presentation Databases
- ▶ Personalized Web Search
- ▶ Social Networks
- ▶ Text Mining
- ▶ XML

Recommended Reading

1. Hatzivassiloglou V., Gravano L., and Maganti A. An investigation of linguistic features and clustering algorithms for topical document clustering. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 224–231.
2. Linden G. People who read this article also read. . . . IEEE Spectr., 45(3):46–60, March 2008.
3. McKeown K., Barzilay R., Chen J., Elson D., Evans D., Klavans J., Nenkova A., Schiffman B., and Sigelman S. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. In Proc. Human Language Technology Conf., 2002, pp. 280–285.

Eleven Point Precision-recall Curve

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

The *11-point precision-recall curve* is a graph plotting the interpolated precision of an information retrieval (IR) system at 11 standard recall levels, that is, {0.0,0.1,0.2,...,1.0}. The method for interpolation is detailed below. The graph is widely used to evaluate IR systems that return ranked documents, which are common in modern search systems.

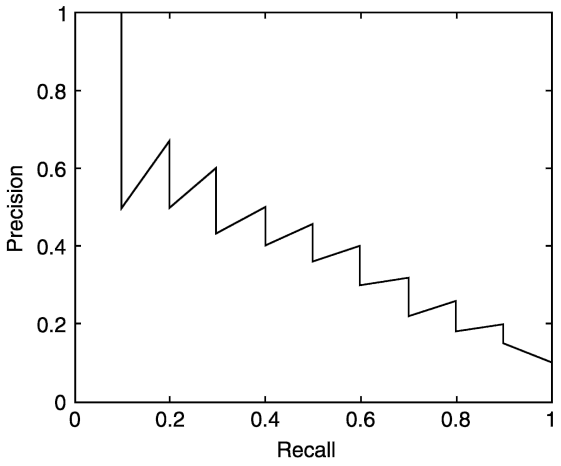
Key Points

In a precision-recall graph, precision is plotted as a function of recall. Assume a search system has retrieved n documents. Then for each $k \in [0,n]$, examine the set of top k retrieved documents and calculate the precision and recall for the set. These values are then plotted on a graph of precision versus recall. Figure 1 shows a typical precision-recall curve. It has a saw-toothed shape because the recall remains the same as k while the precision drops when the $(k + 1)$th retrieved document is not relevant.

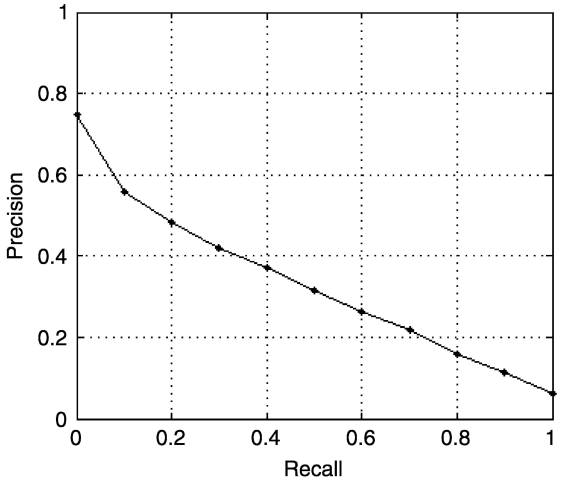
With the original precision-recall graph, averaging the curve over different queries is not straightforward. A standard approach is to plot interpolated precisions at 11 recall levels, namely, 0.0,0.1,...,1.0. The interpolated precision p_{interp} at recall level r is defined as the highest precision for any recall level $r' \geq r$:

$$p_{interp}(r) = \max_{r' \geq r} p(r')$$

Figure 2 shows the interpolated 11-point precision-recall curve for a good system in TREC 8.



Eleven Point Precision-recall Curve. Figure 1.
Uninterpolated precision-recall curve.



Eleven Point Precision-recall Curve. Figure 2.
Interpolated 11-point precision-recall curve.

Cross-references

- ▶ Precision
- ▶ Recall
- ▶ Standard Effectiveness Measures

Embedded Networked Sensing

- ▶ Sensor Networks

Emergent Semantics

Philippe Cudré-Mauroux
Massachusetts Institute of Technology, Cambridge, MA, USA

Synonyms

Bottom-up semantics; Evolutionary semantics

Definition

Emergent semantics refers to a set of principles and techniques analyzing the evolution of decentralized semantic structures in large scale distributed information systems. Emergent semantics approaches model the semantics of a distributed system as an ensemble of relationships between syntactic structures. They consider both the representation of semantics and the discovery of the proper interpretation of symbols as the result of a self-organizing process performed by distributed agents exchanging symbols and having utilities dependent on the proper interpretation of the symbols. This is a complex systems perspective on the problem of dealing with semantics.

Historical Background

Syntax is classically considered as the study of the rules of symbol formation and manipulation [8]. Despite its wide usage in many contexts, the notion of *semantics* often lacks a precise definition. As a least common denominator, it can be characterized as a relationship or mapping established between a syntactic structure and some domain. The syntactic structure is a set of symbols that can be combined following syntactic rules. The possible domains that are related to the symbols may vary. In linguistics, such domains are typically considered as domains of conceptual interpretations. In mathematical logic, a semantic interpretation for a formal language is specified by defining mappings from the syntactic constructs of the language to an appropriate mathematical model. Denotational semantics applies this idea to programming languages. Natural language semantics classically concerns a triadic structure comprising a *symbol* (how some idea is expressed), an *idea* (what is abstracted from reality) and a *referent* (the particular object in reality) [9].

Emergent semantics expresses semantics through purely syntactic, recursive domains. The notions

underlying emergent semantics are rooted in computational linguistics works relating semantics to the analysis of syntactic constructs. In his seminal work on syntactic semantics [10,11], William J. Rapaport defines semantic understanding as the process of understanding one domain in terms of another – antecedently understood – domain. This further raises the question of how the antecedently domain is itself understood. In the same vein, the antecedently understood domain has to be understood in terms of yet another domain, and so on and so forth recursively. To avoid to ground the recursion to a hypothetical base domain, Rapaport suggest the notions of *semantics as correspondence*, and lets the semantic interpretation function recursively map symbols to themselves or to other symbols. By considering the union of the syntactic and semantic domains, Rapaport regards semantics as syntax, i.e., turns semantics into the study of relations within a single domain of symbols and their interrelations. A dictionary is a simple example of a construct based on that paradigm, where the interpretations of symbols (i.e., words) are given by means of the same symbols, creating a closed correspondence continuum. Emergent semantics applies the conception of a closed correspondence continuum to the analysis of semantics in distributed information systems, by promoting recursive analyses of syntactic constructs – such as schemas, ontologies or mappings – in order to capture semantics.

Foundations

Beyond its implication in linguistics – where it is conjectured that human beings inevitably understand meaning in terms of syntactic domains – emergent semantics is considered as being mostly relevant to computer science. Programs, database schemas, models, or ontologies have no capacity (yet) to refer to reality. However, they have various mechanisms at their disposal for establishing relationships between internal symbols and external artifacts. In the settings where humans provide semantics, relationships among symbols – such as constraints in relational databases – are means to express semantics. In order to rectify some of the problems related to the implicit representation of semantics relying on human cognition, some have proposed the use of an explicit reference system for relating sets of symbols in a software system. Ontologies serve this purpose: an ontology vocabulary consists of formal, explicit but partial definitions of the intended meaning of a domain of discourse. In addition, formal constraints (e.g., on the mandatoriness or cardinality of relationships between concepts) are added to reduce the fuzziness of the informal definitions. Specific formal languages (such as OWL) allow to define complex notions and support inference capabilities. In that way, explicitly represented semantics of syntactic structures in an information system consist of relationships between those syntactic structures and some generally agreed-upon syntactic structure. Thus, the semantics is itself represented by a syntactic structure.

In a large scale distributed environment of information agents, such as in the Semantic Web or Peer-to-Peer systems, the aim is to have the agents interoperate irrespective of their initial vocabularies. To that aim, an agent has to map its vocabulary (carrying the meaning as initially defined in its *base* schema or ontology) to the vocabulary of other agents with which it wants to interoperate. Hence, a relationship between local and distant symbols is established. This relationship may be considered as another form of semantics, independent of the initial semantics of the symbols. Assuming that autonomous agents acquire vocabulary terms through relationships to other agents and that agents interact without human intervention, the original *human assigned* semantics would loose its relevance; from an agent's perspective, *new* semantics would then result from the relationships to its environment. This is a novel way of providing semantics to symbols of agents relative to the symbols of other agents with which they interact. Typically, this type of semantic representation is distributed such that no agent holds a complete representation of a generally agreed-upon semantics.

From a global perspective, considering a society of autonomous agents as one system, one can observe that the agents form a complex, self-referential and dynamic system. It is well-accepted that such systems often result in global states, which cannot be properly characterized at the level of local components. This phenomenon is frequently characterized by the notion of *self-organization*. Thus, emergent semantics is not only a local phenomenon, where agents obtain interpretations locally through adaptive interactions with other agents, but also a global phenomenon, where global semantics emerge from the society of agents and represent the common, current *semantic agreement* in the system. This view of semantics as the emergence of a distributed structure from a set of

dynamic processes – or more specifically as some equilibrium state of such processes – is in-line with the generally accepted definitions of emergence and emergent structures in the complex systems literature [2]. In that respect, emergent semantics can be related to dynamic systems disciplines such as evolutionary game theory, semiotic dynamics [13], or graph evolution.

Key Applications

Multimedia Systems: Distributed multimedia applications were the first information systems to take advantage of emergent semantics principles in order to capture the semantics of shared objects. In this context, Santini et al. [12] argue that images do not have an intrinsic meaning, but that they are endowed with a meaning by placing them in the context of other images. From this observation, they propose a system where users are able to manipulate relations between images, and where the semantics of an image is emergent, in the sense that it is a product of the dual activities of users manipulating sets of images and of the database system. Along the same lines, Grosky et al. [5] focus on the role of context for giving meaning to a work of art. In their work, document semantics emerge through the analysis of users' browsing paths through a multimedia collection. They categorize the semantics of each item based on the collection of browsing paths.

Tagging Portals: Emergent semantics can be seen as a natural paradigm to analyze the distributed, user-driven process of giving semantics to items through the use of tags. Extreme Tagging [14] is a technique promoting the tagging of tags and the analysis of the relations between tags. Yeung et al. [15] discuss how shared documents acquire meaning through their associations with other elements. In particular, they demonstrate how different meanings of ambiguous tags can be discovered through the analysis of a tripartite graph involving users, tags, and resources. Herschel et al. [6] discuss query expansion techniques by aggregating users opinions as tags and discuss the role of pragmatics in collaboratively creating semantics.

Heterogeneous Information Systems: Emergent semantics has been suggested as a way to capture semantics in decentralized and heterogeneous information systems [4]. In contrast to mediated integration architectures, recent decentralized integration architectures – such as Peer Data Management Systems – do not require the definition of any global schema or ontology. Thus, the global semantics of such systems can only be captured by considering the collection of conceptualizations as defined by the local databases, along with their interrelations. Semantic Gossiping [1] analyzes transitive closures of schema mappings in that context in order to infer semantic agreement. Related techniques suggest the use of graph theory or probabilistic networks [3] in order to capture global semantics in similar environments. More broadly speaking, emergent semantics techniques are increasingly being seen as a way to minimize manual input and maintenance when dealing with complex and heterogeneous information systems [7].

Cross-references

- ▶ Data Integration
- ▶ Meta Data Management System
- ▶ Multimedia Databases
- ▶ Peer Data Management System
- ▶ Peer-to-Peer Data Integration
- ▶ Schema Mapping

Recommended Reading

1. Aberer K., Cudré-Mauroux P., and Hauswirth M. The Chatty Web: emergent semantics through gossiping. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 197–206.
2. Bar-Yam Y. Dynamics of Complex Systems. Perseus Books Group, 1997.
3. Cudré-Mauroux P. Emergent Semantics: Rethinking Interoperability for Large-Scale Decentralized Information Systems. Ph.D. thesis, 3690, EPFL, Switzerland, 2006.
4. Cudre-Mauroux P. et al. Viewpoints on Emergent Semantics. J. Data Seman., VI:1–27, 2006.
5. Grosky W., Sreenath D., and Fotouhi F. Emergent Semantics and the Multimedia Semantic Web. ACM SIGMOD Rec., 31(4):54–58, 2002.
6. Herschel S., Heese R., and Bleiholder J. An Architecture for Emergent Semantics. In Proc. 15th Int. Conf. on Conceptual Modeling, 2006, pp. 425–434.
7. Howe B., Tanna K., Turner P., and Maier D. Emergent semantics: towards self-organizing scientific metadata. In Proc. Int. Conf. on Semantics of a Networked World, 2004, pp. 177–198.
8. Morris C. Foundations of the theory of signs. International Encyclopedia of Unified Science, 1(2), 1938.
9. Ogden C. and Richards I. The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism. 10th edn., Routledge & Kegan Paul, London, 1923.
10. Rapaport W. Syntactic semantics: foundations of computational natural-language understanding. In Aspects of Artificial Intelligence, Kluwer Academic, Dordrecht, Holland, 1988, pp. 81–131.
11. Rapaport W. What Did You Mean by That? Misunderstanding, Negotiation, ans Syntactic Semantics. Minds and Mach., 13(3):397–427, 2003.

12. Santini S., Gupta A., and Jain R. Emergent Semantics through Interaction in Image Databases. IEEE Trans. Knowl. Data Eng., 13(3):337–351, 2001.
13. Steels L. Semiotic dynamics for embodied agents. IEEE Intelligent Systems, 21(3):32–38, 2006.
14. Tanasescu V. and Streibel O. Extreme tagging: emergent semantics through the tagging of tags. In Proc. Int. Workshop on Emergent Semantics and Ontology Evolution, 2007, pp. 84–94.
15. Yeung C., Gibbins N., and Shadbolt N. Understanding the semantics of ambiguous tags in folksonomies. In Proc. Int. Workshop on Emergent Semantics and Ontology Evolution, 2007, pp. 108–121.

Emerging Pattern Based Classification

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Synonyms

Contrast pattern based classification

Definition

The term "emerging-pattern based classification" refers to any classification algorithm that uses emerging patterns to directly build classifiers or to help build/improve other classifiers.

Key Points

The first approach to consider emerging-pattern based classification is CAEP (classification by aggregating emerging patterns) [2]. The main idea is to aggregate (sum) the discriminating power of many of the emerging patterns contained in a case to be classified. The discriminating power of an emerging pattern is often reflected in the support difference of the pattern in the opposing classes. For each class, the emerging patterns of that class contained in the case are aggregated to form a score; the class with the highest score is deemed to be the class of the case. Score normalization can be used to deal with data/battern imbalance between classes. This classification method can lead to high quality classifiers, comparable or better than other classifiers.

A second major direction in emerging-pattern based classification is the DeEPs method [2], which is an instance-based classification method using emerging patterns. After a case to be classified is given, DeEPs will compute the jumping emerging patterns contained in this case from the training data, and use the training data (that contain some of the discovered jumping emerging patterns) to compute the aggregated scores for the case and the classes. The DeEPs method avoids a deficiency of getting duplicate discriminating power contribution from near equivalent emerging patterns.

Recently, many other emerging-pattern based classification methods have been introduced (e.g., [1]), and several papers study the noise tolerance of emerging pattern based classifiers. More references can be found at www.cs.wright.edu/~gdong/EPC.html.

Cross-references

▶ Applications of Emerging Patterns for Microarray Gene Expression Data Analysis
▶ Associative Classification
▶ Classification
▶ Classification by Association Rule Analysis
▶ Emerging Patterns

Recommended Reading

1. Alhammady H. and Ramamohanarao K. Using emerging patterns to construct weighted decision trees. IEEE Trans. Knowl. Data Eng., 18(7):865–876, 2006.
2. Dong G., Zhang X., Wong L., Li J. CAEP: classification by aggregating emerging patterns. Discov. Sci., 30–42, 1999.
3. Li J., Dong G., Ramamohanarao K., and Wong L. DeEPs: a new instance-based lazy discovery and classification system. Mach. Learn., 54(2):99–124, 2004.

Emerging Patterns

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Synonyms

Contrast pattern; Group difference

Definition

Roughly speaking, emerging patterns [3] are patterns whose support changes significantly from one dataset to another. The definition below captures significant change in terms of big growth rate.

Formally, let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two datasets, and let X be an itemset. For each integer $i \in \{1, 2\}$, let $supp_i(X)$ denote the support of X in $\mathcal{D}_i$. The *growth rate* of X's support from $\mathcal{D}_1$ to $\mathcal{D}_2$ is defined as

$$\text{GrowthRate}(X) = \begin{cases} 0 & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) = 0 \\ \infty & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) \neq 0 \\ \frac{supp_2(X)}{supp_1(X)} & \text{otherwise} \end{cases}$$

Observe that growth rates can also be defined in terms of counts instead of supports. One can also adjust the definition to avoid division by zero, by letting $\text{GrowthRate}(X) = \frac{supp_2(X)}{supp_1(X)+\epsilon}$ for some fixed $\epsilon > 0$.

Given a growth-rate threshold $\rho > 1$, an itemset X is called a *ρ-emerging pattern* (or simply *emerging pattern*, or *EP*) from $\mathcal{D}_1$ to $\mathcal{D}_2$ if $\text{GrowthRate}(X) \geq \rho$; the dataset $\mathcal{D}_1$ is called the *background* dataset, and $\mathcal{D}_2$ the *target* dataset, of X. Emerging patterns whose growth rate is ∞ are called *jumping emerging patterns* (JEPs).

As with frequent itemsets, the number of emerging patterns can be large. One common approach to overcome this problem is to focus on the minimal emerging patterns with respect to the set containment relation.

Historical Background

The concept of emerging patterns was motivated to capture emerging trends or change patterns over time, or significant differences/contrasts between datasets or data classes. The mining of emerging patterns can be viewed as a special case of comparative data mining, where one compares several datasets against each other in order to discover similarities and contrasts, etc. among the datasets. Emerging patterns were initially defined over relational or transactional data [3] and were recently generalized to sequences [6] and graphs [13], etc. In addition to emerging patterns, other patterns on change/difference, together with their mining, were also considered in many other studies [2,3,11,14,16].

Several algorithms have been proposed to mine emerging patterns, including the border-differential algorithm [3], a constraint-based bounding algorithm, a tree-based algorithm, a minimal hypergraph transversal based algorithm [1], a ZBDD based algorithm [12], and an incremental maintenance algorithm [9]. A theoretical analysis on the complexity of emerging pattern mining was reported in [16], and a study on the relationship between emerging patterns and rough set reducts was reported in [14]. Reference [10] proves that the space of jumping emerging patterns is convex and linked that space to version spaces. Reference [8] gives an efficient algorithm for mining minimal (with respect to set containment) emerging patterns, and linked the mining of emerging patterns to the mining of minimal generators and closed patterns. The mining of emerging patterns was also studied in several papers whose focus are on classification using emerging patterns (cf www.cs.wright.edu/~gdong/EPC.html).

The study of emerging patterns led to two main directions of extensive and fruitful research, namely the use of emerging patterns in classification, and the application of emerging patterns for microarray gene expression data analysis.

Foundations

The border-differential algorithm [3] computes the border of certain sets of emerging patterns, by performing border differential. Borders [3] represent, in a concise manner, large convex collections of itemsets, including certain collections of emerging patterns. A border contains two boundary collections of itemsets, namely a *minB* and a *maxB*; it represents the convex set $\{z \mid \exists x \in minB, \exists y \in maxB \text{ such that } x \subseteq z \subseteq y\}$ of itemsets. For example, $<\{\{2\}, \{3, 4\}\}, \{\{2, 3, 4\}\} >$ is a border, where $\{\{2\}, \{3, 4\}\} = minB$ (containing two itemsets) and $\{\{2, 3, 4\}\} = maxB$ (containing one itemset), representing $\{\{2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{2, 3, 4\}\}$. The main steps of the border-differential algorithm are some iterative expansion–minimization steps used to find the minimal itemsets which are contained in a given positive transaction but which are not contained in any of given negative transactions. The border-differential algorithm can directly compute the set of all jumping emerging patterns, and the set of all emerging patterns whose support in one dataset is lower than a threshold and whose support in the other dataset is higher than another threshold. Experiments show that the algorithm was fairly fast and can handle datasets of up to 75 attributes.

Reference [9] studies how to incrementally modify and maintain the concise border descriptions of the collection of all jumping emerging patterns when small changes to the original data occur. It introduces

algorithms to handle four types of changes: insertion of new data, deletion of old data, addition of new attributes, and deletion of old attributes. Similar to the border-differential algorithm, the incremental algorithms obtain the new borders by manipulating the old borders. Experiments show that the incremental algorithms are much faster than the computing-from-scratch method.

Reference [1] shows that computing minimal jumping emerging patterns is equivalent to the well known problem of enumerating the minimal transversals of a hypergraph. It proposes a bottom up approach for computing minimal hypergraph transversals that is particularly suited to the types of hypergraphs arising in jumping emerging pattern mining. Earlier, the three authors of [1] also proposed a tree based algorithm to mine emerging patterns; the tree stores two (or more) counts for each node, one for each dataset; two item orderings, one based on frequency ratio and one based on frequency, are used; it was found that the algorithm was faster than earlier ones and that minimizing tree size may not lead to the shortest computation time.

Reference [12] introduces an efficient algorithm for mining emerging patterns (and generalizations). The algorithm is based on the so-called zero suppressed binary decision diagrams (ZBDDs). Binary decision diagrams (BDDs) are directed acyclic graphs which are efficient representations of boolean formulae. A ZBDD is a special type of BDD, originally introduced for set-manipulation in combinatorial problems. In [12], ZBDD is used to compress sparse high dimensional itemsets/data, and to allow the reuse of past computations. This algorithm can be hundreds of times faster than the tree-based algorithm, and can successfully mine emerging patterns from data with several hundreds (or even thousands) of attributes.

Key Applications

Emerging patterns are useful for capturing multi-dimensional contrasts between datasets/classes. They have been extensively used in classification, and for microarray gene expression data analysis (especially for cancers). Emerging patterns can also be used (i) to discover emerging trends in temporal databases by comparing datasets collected from two time intervals, (ii) to identify rare events, and (iii) to detect network intrusion. References on these can be found at www.cs.wright.edu/~gdong/EPC.html.

Cross-references

▶ Applications of Emerging Patterns for Microarray Gene Expression Data Analysis
▶ Emerging Pattern Based Classification

Recommended Reading

1. Bailey J., Manoukian T., and Ramamohanarao K. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 485–488.
2. Bay S.D. and Pazzani M.J. Detecting change in categorical data: mining contrast sets. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 302–306.
3. Dong G., Li J. Efficient mining of emerging patterns: discovering trends and differences. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 43–52. Journal version [5].
4. Dong G., Han J., Lam J.M.W., Pei J., Wang K., and Zou W. Mining constrained gradients in large databases. IEEE Trans. Knowl. Data Eng., 16(8):922–938, 2004.
5. Dong G. Li J. Mining border descriptions of emerging patterns from dataset pairs. Knowl. Inf. Syst. 8(2):178–202, 2005.
6. Ji X., Bailey J., and Dong G. Mining minimal distinguishing subsequence patterns with gap constraints. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 194–201. Journal version [7].
7. Ji X., Bailey J., and Dong G. Mining Distinguishing Subsequences Patterns with Gap Constraints. Knowl. Inf. Syst. 11 (3):259–289, 2007.
8. Li J., Liu G., and Wong L. Mining statistically important equivalence classes and δ-discriminative emerging patterns. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 430–439.
9. Li J., Manoukian T., Dong G., and Ramamohanarao K. Incremental maintenance on the border of the space of emerging patterns. Data Min. Knowl. Discov. 9(1):89–116, 2004.
10. Li J., Ramamohanarao K., and Dong G. The space of jumping emerging patterns and its incremental maintenance algorithms. In Proc. 17th Int. Conf. on Machine Learning: 2000, pp. 551–558.
11. Liu B., Hsu W., and Ma Y. Discovering the set of fundamental rule changes. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 335–340.
12. Loekito E. and Bailey J. Fast Mining of High Dimensional Expressive Contrast Patterns Using Zero-Suppressed Binary Decision Diagrams. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 307–316.
13. Ting M.H.T. and Bailey J. Mining minimal contrast subgraph patterns, In Proc. SIAM International Conference on Data Mining, 2006, pp. 638–642.
14. Terlecki P. and Walczak K. On the relation between rough set reducts and jumping emerging patterns. Inf. Sci., 177(1):74–83, 2007.
15. Vreeken J., van Leeuwen M., and Siebes A. Characterising the difference. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 765–774.

16. Wang L., Zhao H., Dong G., and Li J. On the complexity of finding emerging patterns. Theor. Comput. Sci. 335(1):15–27, 2005.
17. Webb G.I., Butler S.M., and Newlands D.A. On detecting differences between groups. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 256–265.

Encryption

► Data Encryption

End User

► Actors/Agents/Roles

Ensemble

Zhi-Hua Zhou
Nanjing University, Nanjing, China

Synonyms

Committee-based learning; Multiple classifier system; Classifier combination

Definition

Ensemble is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn *one* hypothesis from training data, ensemble methods try to construct a *set* of hypotheses and combine them.

Historical Background

It is difficult to trace the starting point of the history of ensemble methods since the basic idea of deploying multiple models has been in use for a long time. However, it is clear that the hot wave of research on ensemble methods since the 1990s owes much to two works. The first is an applied research conducted by Hansen and Salamon at the end of 1980s [5], where they found that predictions made by the combination of a set of neural networks are often more accurate than predictions made by the best single neural network. The second is a theoretical research conducted in 1989, where Schapire proved that *weak learners* which are slightly better than random guess can be boosted to *strong learners* that are able to make very accurate predictions, and the proof resulted in Boosting, one of the most influential ensemble methods [10].

Foundations

Terminologies

Learners composing an ensemble are usually called *base learners*. Many ensemble methods are able to boost *weak learners* which are slightly better than random guess to *strong learners* which can make very accurate predictions. So, "base learners" are also referred as "weak learners". However, it is noteworthy that although most theoretical analyses work on weak learners, base learners used in practice are not necessarily weak since using not-so-weak base learners often results in better performance.

Base learners are usually generated from training data by a *base learning algorithm* which can be decision tree, neural network or other kinds of machine learning algorithms. Most ensemble methods use a single base learning algorithm to produce *homogeneous* base learners, but there are also some methods which use multiple learning algorithms to produce *heterogeneous* learners. In the latter case there is no single base learning algorithm and thus, some people prefer calling the learners *individual learners* or *component learners* to "base learners", while the names "individual learners" and "component learners" can also be used for homogeneous base learners.

Methods

Typically, an ensemble is constructed in two steps. In the first step, a number of base learners are produced. Here the base learners can be generated in a *parallel* style, or in a *sequential* style where the generation of a base learner has influence on the generation of subsequent learners. In the second step, the base learners are combined to use. The most popular combination scheme for classification is *majority voting*, while the most popular combination scheme for regression is *weighted averaging*. The employment of different base learner generation processes and/or different combination schemes leads to different ensemble methods.

The following three paragraphs briefly describe the working routines of three representative ensemble methods, *Boosting* [10], *Bagging* [2] and *Stacking* [13], respectively. Here, binary classification is considered

for simplicity. That is, let $\mathcal{X}$ and $\mathcal{Y}$ denote the instance space and the set of class labels, respectively, assuming $\mathcal{Y} = \{-1, +1\}$. A training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ is given, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ $(i = 1, \ldots, m)$.

For Boosting, let us consider the most famous algorithm, AdaBoost, as an example. In the first step, a number of base learners are produced by emphasizing each learner on the training examples that are wrongly predicted by preceding learners. Here, a weight distribution is maintained over the training examples, which is initialized by equal weights. In the tth learning round, a base learner $h_t : \mathcal{X} \to \mathcal{Y}$ is generated from the training set by using the current weight distribution, D_t. Then, h_t is evaluated on the training set and let ε_t denote the error. The weight distribution for the next learning round, D_{t+1}, is generated in the way that for the training example (x_i, y_i), $D_{t+1}(i) = \frac{D_t(i)}{Z_t} exp\left(\frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) y_i h_t(x_i)\right)$ where Z_t is a normalization factor which enables D_{t+1} to be a distribution. In the second step, the base learners h_t's $(t = 1, \ldots, T)$ are combined by $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

For Bagging, in the first step a number of base learners are trained from *bootstrap samples*. A bootstrap sample is obtained by subsampling the training set with replacement, where the size of a sample is as the same as that of the training set. Thus, for a bootstrap sample, some training examples may appear but some may not, where the probability that an example appears at least once is about 0.632. On each sample a base learner $h_t : \mathcal{X} \to \mathcal{Y}$ is produced by calling a base learning algorithm. In the second step, Bagging combines the learners h_t's $(t = 1, \ldots, T)$ by majority voting, i.e., $H(x) = argmax_{y \in \mathcal{Y}} \sum_{t=1}^{T} 1(y = h_t(x))$ where the value of $1(a)$ is 1 if a is *true* and 0 otherwise.

For a typical implementation of Stacking, in the first step, a number of individual learners, $h_t : \mathcal{X} \to \mathcal{Y}$ $(t = 1, \ldots, T)$, are generated from the training set by employing different learning algorithms. In the second step, the individual learners are combined by using another learner. Here, for the training example (x_i, y_i), a corresponding example (z_i, y_i) is produced, where $z_i = (h_1(x_i), \ldots, h_T(x_i))$. Then, from $\{(z_i, y_i)\}$ $(i = 1, \ldots, m)$ a learner $F : h_t^T \to \mathcal{Y}$ is generated by calling a learning algorithm, which is used to combine the individual learners by $H(x) = F(h_1(x), \ldots, h_T(x))$.

The above methods have many variants. For example, *Random Forests* [3], which has been deemed as one of the most powerful ensemble methods, is a variant of Bagging.

There are many other established ensemble methods, e.g., *Random Subspace* [6]. This method generates a number of base learners from different *subspaces* of the training set in the first step, and then combines these base learners via majority voting in the second step. Here, a subspace is actually a subset of the original attribute set.

It is worth mentioning that in addition to classification and regression, ensemble methods have also been designed for clustering [11] and other kinds of machine learning tasks.

It was thought that using more base learners will lead to a better performance, yet Zhou et al. [15] proved the "many could be better than all" theorem which indicates that this may not be the fact. It was shown that after generating a set of base learners, selecting some base learners instead of using all of them to compose an ensemble is a better choice. Such ensembles are called *selective ensembles*.

Note that usually the computational cost for building an ensemble comprising T base learners is roughly T times the cost of training a single learner. So, from the view of computational complexity, training an ensemble is almost as efficient as training a single learner.

Why Useful?

The generalization ability of an ensemble is usually much stronger than that of a single learner, which makes ensemble methods very attractive. An important question is, why can the ensembles be superior to single learners? For this, Dietterich [4] gave three reasons by viewing the nature of machine learning as searching a hypothesis space for the most accurate hypothesis. The first reason is that, the training data might not provide sufficient information for choosing a single best learner. For example, there may be many learners performing equally well on the training set. Thus, combining these learners may be a better choice. The second reason is that, the search processes of the learning algorithms might be imperfect. For example, even if there exists a unique best hypothesis, it might be difficult to achieve since running the algorithms results in sub-optimal hypotheses. Thus, ensembles can compensate for such imperfect search processes. The third reason is that, the hypothesis space being searched might not contain the true target function, while

ensembles can give some good approximation. For example, it is well-known that the classification boundaries of decision trees are linear segments parallel to coordinate axes. If the target classification boundary is a smooth diagonal line, using a single decision tree cannot lead to a good result but a good approximation can be achieved by combining a set of decision trees. Although these intuitive explanations are reasonable, they lack rigorous theoretical analyses.

The *bias-variance decomposition* is often used in studying the performance of ensemble methods [1,15]. It is known that Bagging can significantly reduce the variance, and therefore it is better to be applied to learners suffered from large variance, e.g., unstable learners such as decision trees or neural networks. Boosting can significantly reduce the bias in addition to reducing the variance, and therefore, on weak learners such as decision stumps, Boosting is usually more effective.

There are many theoretical studies on famous ensemble methods such as Boosting and Bagging, yet it is far from a clear understanding of the underlying mechanism of these methods. For example, empirical observations show that Boosting often does *not* overfit even after a large number of rounds, and sometimes it is even able to reduce the generalization error after the training error has already reached zero. Although many people have studied this phenomenon, theoretical explanations are still in arguing.

Accuracy and Diversity

Generally, in order to construct a good ensemble, the base learners should be as more accurate as possible, and as more diverse as possible. This has been formally shown by Krogh and Vedelsby [7], and emphasized by many other people.

The definition of *accuracy* is clear, and there are many effective processes for estimating the accuracy of learners, such as cross-validation, hold-out test, etc. However, there is no rigorous definition on what is intuitively perceived as *diversity*. Although a number of diversity measures have been designed, Kuncheva and Whitaker [8] revealed that the usefulness of existing diversity measures in building ensembles is suspectable.

In practice, the diversity of the base learners can be introduced from different channels, such as subsampling the training examples, manipulating the attributes, manipulating the outputs, injecting randomness into learning algorithms, or even using multiple mechanisms simultaneously.

Key Applications

Ensemble methods have already been used in diverse applications such as optical character recognition, text categorization, face recognition, medical diagnosis, gene expression analysis, etc. Actually, ensemble methods can be used wherever machine learning techniques can be used.

Future Directions

A serious deficiency of ensemble methods is the lack of comprehensibility, i.e., the knowledge learned by ensembles is not understandable to the user. Improving the comprehensibility of ensembles [14] is an important yet largely understudied direction.

Diversity plays an important role in ensembles, yet currently no diversity measures is satisfying [8]. Exploring the relation between the performance of ensembles and the properties of base learners to design useful diversity measures is an interesting direction, which may lead to the development of more powerful ensemble methods.

Although there are much theoretical analyses on ensemble methods, many underlying mechanisms of successful ensemble methods are not clear. So, more theoretical studies are needed. Another ambitious attempt is to establish a general theoretical framework for ensemble methods.

Experimental Results

Empirical studies on popular ensemble methods have been reported in many papers, such as [1,9,12].

Data Sets

A large collection of datasets commonly used for experiments can be found at http://www.ics.uci.edu/~mlearn/MLRepository.html

Url To Code

The code of *Random Forests* can be found at http://www.stat.berkeley.edu/~breiman/RandomForests/

Cross-references

- ▶ Bagging
- ▶ Boosting
- ▶ Decision Tree

- ► Neural Networks
- ► Support Vector Machine

Recommended Reading

1. Bauer E. and Kohavi R. An empirical comparison of voting classification algorithms: bagging, Boosting, and variants. Mach. Learn., 36(1–2):105–139, 1999.
2. Breiman L. Bagging predictors. Mach. Learn., 24(2):123–140, 1996.
3. Breiman L. Random forests. Mach. Learn., 45(1):5–32, 2001.
4. Dietterich T.G. Machine learning research: Four current directions. AI Magn., 18(4):97–136, 1997.
5. Hansen L.K. and Salamon P. Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell., 12(10):993–1001, 1990.
6. Ho T.K. The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell., 20(8):832–844, 1998.
7. Krogh A. and Neural network ensembles, cross validation, and active learning. In Advances in Neural Information Processing Systems 7, G. Tesauro, D.S. Touretzky, and T.K. Leen (eds.). MIT Press, Cambridge, MA, 1995, pp. 231–238.
8. Kuncheva L.I. and Whitaker C.J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Mach. Learn., 51(2):181–207, 2003.
9. Opitz D. and Maclin R. Popular ensemble methods: An empirical study. J. Artif. Intell. Res., 11:169–198, 1999.
10. Schapire R.E. The Boosting approach to machine learning: An overview. In Nonlinear Estimation and Classification, D.D. Denison, M.H. Hansen, C. Holmes, B. Mallick, and B. Yu (eds.). Springer, Berlin, 2003.
11. Strehl A. and Ghosh J. Cluster ensembles – a knowledge reuse framework for combining multiple partitionings. J. Mach. Learn. Res., 3:583–617, 2002.
12. Ting K.M. and Witten I.H. Issues in stacked generalization. J. Artif. Intell. Res., 10:271–289, 1999.
13. Wolpert D.H. Stacked generalization. Neural Netw., 5(2): 241–260, 1992.
14. Zhou Z.-H., Jiang Y., and Chen S.-F. Extracting symbolic rules from trained neural network ensembles. AI Commun., 16(1):3–15, 2003.
15. Zhou Z.-H., Wu J., and Tang W. Ensembling neural networks: Many could be better than all. Artif. Intell., 137(1–2):239–263, 2002.

Enterprise Application Integration

Janette Wong
IBM Canada Ltd, ON, Canada

Synonyms

EAI; Application-to-application integration; Application-centric interfacing

Definition

Enterprise Application Integration (EAI) is concerned with making applications work together seamlessly, by sharing data and functions among data sources and applications across heterogeneous platforms, facilitated by the use of common middleware. The applications can be a mixture of in-house developed, commercially packaged, and open-source applications; some are contemporary applications, some are legacy systems. Some of the applications were not designed to work together originally. The majority of the definitions set the boundary of EAI for integration within an enterprise. Integration across enterprises involves additional and sometimes different considerations that are covered by solutions that fall under the category of business-to-business (B2B) integration [10]. Variations of the definition of EAI can be found in existing literature. Some of the definitions include the idea that the ultimate goal of EAI is to enable the creation of new business solutions.

Historical Background

Integration of applications started with the sharing of data using files. An application outputs data to a file with specific file naming and placement conventions so that other applications can retrieve the contents of the file. Important decisions include deciding on the file format and how frequently to produce and consume the file. A common issue is that applications can get out of synchronization. When an application uses obsolete data, resulting consequences can be severe.

The next step in the evolution of integration came when client-server architectures became popular and gave rise to the notion of Application Programming Interfaces (APIs) through which a client can invoke the services provided by a remote server. The underlying remote invocation is handled by Remote Procedure Calls (RPC). The main disadvantage is that the API that is being invoked must be in a language and platform compatible with the calling application, which implies multiple APIs are needed to expose an application to other applications on different platforms and languages.

With the advent of object-oriented programming, distributed components in the form of objects allow applications to interact irrespective of their locations and platforms. Common Object Request Broker Architecture (CORBA) took the RPC ideas further by

applying them to an object-oriented environment. Although the status of CORBA being the standard distributed object platform has been replaced by more recent platforms such as Java 2 Platform Enterprise Edition (J2EE) and .NET, the ideas from CORBA are still being used today.

When using RPC and distributed objects, applications invoke the APIs of each other directly. This is known as *point-to-point* integration and it has some drawbacks which are aggravated as the number of applications to be integrated increases. Point-to-point integration results in tightly coupled applications and is not a scalable approach.

In the 1990s, ERP (Enterprise Resource Planning) systems became popular, providing a variety of business functions such as accounting and inventory management. Enterprises want to leverage existing functions and data in the ERP systems, but it was difficult to integrate them with other applications. Vendors responded by providing *connectors* (also called *adapters*) to the ERP systems. As the number of applications to be integrated increased, it became clear that a common infrastructure which provides services to allow the applications to interact and manages those interactions would be useful. Such an infrastructure would also minimize the dependencies and assumptions applications make about each other. Industry analysts created the term EAI to represent the software in this infrastructure which enables and facilitates integration.

The success of the Internet provides new opportunities to develop strategic business solutions that bring competitive advantages and fuel the need for integration even further. Standards such as Extensible Markup Language (XML) and industry-specific business protocols provide more common basis for integration. The shift to service-oriented integration, supported by technologies such as Web Services, SOA (Service Oriented Architecture), and SCA (Service Component Architecture) have now taken integration into a new generation.

Foundations

Integrating a diverse set of applications is a big challenge. Following are typical business and technological issues involved in EAI:

- The mindset and policies of organizations need to adapt to an integrated environment. Traditionally, an IT department has almost full control over the applications for which it is responsible. In an integrated environment, an application is only one part of a bigger system. Successful integration requires the cooperation between business units and organizations in addition to cooperation between applications.
- Considerable pressure and expectation are placed on integrated systems to operate seamlessly and optimally, even under adverse and unexpected conditions. This is the result of many integrated systems providing vital business functions where impact, such as cost and reputation, to the business when a failure occurs can be severe.
- The number of applications to be integrated can range from tens to hundreds, or even thousands in a large enterprise.
- Applications to be integrated comprise of legacy systems running on mainframes (e.g., CICS and MVS applications), Commercial-Off the Shelf (COTS) applications such as SAP R/3 and PeopleSoft, in-house developed applications, database management systems, corporate directories, and open-source applications. Such a diverse set of applications often use different technologies and run on different platforms; use different architectural styles: monolithic, 2-tiers, and 3-tiers. Data format and semantics among the applications are usually different. This is known as *semantic dissonance*. Qualities of services among the applications may also be different and need reconciliation.
- Applications, particularly the older ones, were not originally designed to be integrated with.
- Applications are often autonomous: they have different owners and will continue to evolve independently.
- Applications are often geographically distributed which requires remote communication. Networks can be slow, unreliable, and insecure. The applications themselves may also be unavailable at times.
- Applications cannot be changed for the purpose of integration due to organizational and technical reasons. If changes are needed, they must be minimized.
- The breadth of business knowledge and technologies involved require a broad set of skills that is difficult to find in a small number of individuals or in a single organization.

To overcome the technical issues, EAI solutions employ a number of techniques:

- Use the design principle of *loose coupling* to minimize assumptions and dependencies applications have on each other. This will allow applications to evolve independently and if one application changes or an exception occurs, impact to other applications is minimal.
- Use connectors to adapt applications to the target integrated environment. Connectors provide a variety of services including mapping interfaces, transforming data, and exception handling. Performance and reusability are two important aspects in the design of connectors. Many reusable and configurable connectors for a variety of applications and legacy systems are available from vendors.
- Use a middleware integration layer to manage interactions among the applications. This middleware layer runs on a distributed object technology platform and uses a hub and spoke or a Message Bus [1] (also known as Enterprise Service Bus (ESB)) architecture. These architectural styles are highly scalable. Applications only need to be connected to this middleware integration layer, point-to-point integration is no longer needed.
- The middleware layer supports synchronous interactions and *asynchronous messaging*, provides message transformation and routing services, supports distributed transactions and provides security services.
- Asynchronous messaging is one of the techniques to achieve loose coupling. With asynchronous messaging, an application sends a message to a message queue or a Message Channel [1] and the application continues its own processing. There is no need for the sending application to stop and wait for the receiving application to receive the message and respond. This is known as *non-blocking* communication. The message is persisted by the Message Channel which is responsible for forwarding the message to the receiving application. The messaging system also provides transactional support for messages and handles recovery in the case of failure.
- Message transformation is another technique to achieve loose coupling. Transformation allows the format of data and messages between interacting applications to be different.
- Routing allows for the dynamic determination of the target application that is to be invoked. The source application which initiates the interaction can still have some control over which target application it interacts with, but it no longer has to bear the full responsibility of determining the address of the target application, a process known as *binding*. The middleware supports *dynamic binding* which allows the actual address of a target application to be determined based on factors such as load balancing and availability. Dynamic binding is yet another technique to achieve loose coupling.
- The middleware layer may also include sophisticated servers such as:
  - Process servers with embedded workflow engines to orchestrate or choreograph applications to carry out higher level business processes. Process servers automate the execution of tasks within a business process expressed using a standard language such as Business Process Execution Language (BPEL). Process servers often provide the capability to use business rules to control the execution of business tasks.
  - Database and directory integration servers which federates multiple databases and directory servers.
  - Portal servers which aggregate data from multiple applications to create a unified view and interaction experience for end users.

Although the middleware integration layer provides a great deal of assistance in creating an integration solution, even before using the middleware, integration architects still have much to reconcile between business and technical requirements. One important area of focus is on quality of service. Even when a participating application meets all of the functional and data requirements for integration into a target environment, that application may lack quality of service. Quality of service attributes of an application can include:

- Availability
- Reliability
- Performance
- Whether an application idempotent (i.e., no negative effects if invoked multiple times)
- Whether multiple instances of the application can be invoked concurrently
- Whether an application can participate in a transaction

Approaches to Application Integration

With such a variety of technologies, considerations that are technical and business in nature and span inter and intra enterprises, patterns are good ways of acquiring a high level understanding of application integration, its key concepts, the recurring problems, and recommended solutions. Over the years, approaches such as the ones captured by Adams [2], Trowbridge et al. [3] and Ruh et al. [4] have been documented as a set of application integration patterns.

- *Presentation integration* connects applications to give end users a unified view of the data. Among the three approaches to integration, this is the least complicated. Trowbridge [3] referred to this approach as Portal Integration and elaborated upon the different levels of sophistication, ranging from simple display of data from multiple applications to *cross-pane interactivity* where user actions in one portion of the display can affect another portion of the display. Adams [2] provided a broader coverage of presentation issues by considering access from different types of devices such as voice-enabled devices, personalized delivery of information according to user preferences, and security considerations such as single sign-on.
- *Data integration* connects multiple data stores to give applications a unified view of the data. It is responsible for resolving semantic dissonance among the data stores. The two major approaches to data integration are either physically replicating data from the multiple data stores into a central data store, or combining data from multiple data stores in real time presenting an aggregated but virtual view of the data. Sauter [5,6,7,9] identified a several patterns for data integration including Data Consolidation, Data Federation, and Data Cleansing.
- *Process integration* orchestrates interactions among multiple applications, using the services or subprocesses provided by the applications to perform a higher level business process often involving business rules. As part of process integration, business performance of processes and activities can be monitored to provide data that are vital for day to day business decisions.

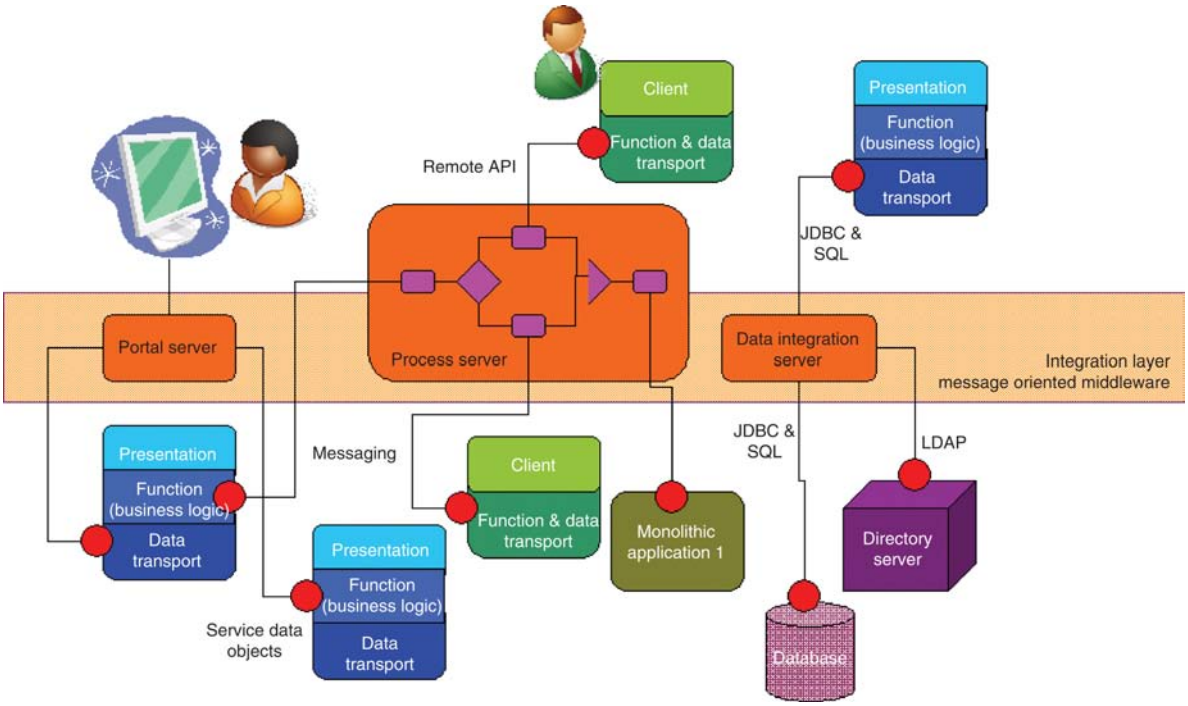
In an integration solution, it is not uncommon to see a combination of the different approaches. Adams [2] additionally provided business patterns to consider the business drivers and requirements before considering the integration approaches.

Figure 1 depicts the different approaches to integration being used together.

Connecting to Applications

Once the overall business purpose of an integration effort is clear and a logical high level understanding of the integration solution has been achieved, another important consideration is how to connect the participating applications to the middleware layer. Many contemporary business applications follow a three-tiered architecture with one layer for presentation, one layer for business logic, and a third layer for handling data. This three-tiered architecture gives rise to three common ways of connecting to an application:

- Connect an application to the middleware layer through the presentation tier of an application. This is the most limiting way of connecting to an application because it is restricted to the data and functions that an application exposes to its end users. It usually is slow in performance and is tightly coupled with the presentation tier of the application and therefore has limited potential for reuse. *Screen scraping* is a common technique to integrate an application through its presentation tier. The integration is achieved by capturing data displayed by the application and extracting relevant information from it.
- Connect an application to the middleware layer through the business logic tier of the application. Connections can be achieved by invoking APIs provided by the application, for instance, by using distributed object technologies, or by invoking the web services interfaces of the application. Asynchronous messaging is used in many cases. Hohpe and Woolf [1] provide an extensive set of asynchronous messaging patterns used for application integration.
- Connect an application to the middleware layer through the data tier of an application. This type of connection mechanism is more flexible and provides access to more data than connecting to the presentation tier. The main disadvantage is that the business logic that resides in the application that processes and manages the data is bypassed, which can lead to data integrity problems. Another disadvantage is that the integration is tied to the data



Enterprise Application Integration. Figure 1. The middleware integration layer connecting various applications (monolithic, 2-tier, 3-tier) and data sources (database, directory server). Different types of integration are supported by the integration layer.

model of the application. Some applications do not consider their data model to be part of their public interface. Directly accessing a "private" data model is a fragile approach. If the data model changes, the integration code has to change accordingly. Connecting to the data tier is more suitable if separate logic, independent of the ones in the existing applications, is intended for processing the aggregated data. A data warehouse is a typical example where logic resides to analyze the aggregated data for the purposes of creating reports and making decisions. The data warehouse typically connects to various data sources through their data tier.

- A variety of tools and data access middleware can be used to connect to an application through the data tier. Such tools and middleware connect to the databases using standard APIs such as Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC). Often, the tools and middleware provide data transformation as well.

Among the three ways of connecting to an application, connecting to the business logic tier is the most flexible and solves the widest range of integration problems, and has the greatest reuse potential. When new business logic and in particular, workflow and transactional integrity, are involved, connecting to the business logic tier is the most appropriate. However, connecting to the business logic tier is also the most complicated and requires the greatest set of skills. Connecting to the data tier directly will always be useful where the situations call for direct connection to data sources and bypassing existing business logic is not an issue.

It should not be surprising that the three major approaches to integration have affinity to the different ways of connecting to an application. The presentation integration approach is typically combined with connecting to an application via its data or presentation tier. The data integration approach is most often associated with connecting to an application via its data tier. Finally, the process integration approach is typically achieved by connecting to an application via its business logic tier.

Standards for Application Integration

While EAI is often interpreted as application integration within an enterprise, with the need for application integration outside of an enterprise, with supply chain,

business partners, and regulatory agencies, many organizations want a consistent strategy and approach to integration, irrespective of whether the integration is for inter or intra enterprise. Vendors support this by having their offerings designed to support both types of integration, using the same set of tools and concepts, maximizing reuse while avoiding duplication. Many tools support standards that are applicable to integration. Chari and Seshadri [8] proposed a framework for organizing and navigating the standards according to three orthogonal dimensions. Each dimension is divided into multiple categories.

- *Application architecture*: This dimension classifies a standard according to the architectural layer of an application to which the standard applies. Three architectural layers comprise this dimension: presentation logic, business logic, and data logic.
- *Integration level*: This dimension classifies a standard according to the level of integration to which the standard applies. Three levels of integration comprise this dimension: transport, data format, and process.
- *Industry domain specificity*: This dimension classifies a standard according to whether the standard is applicable to a variety of industries or to a particular industry.

Some standards are sufficiently broad in scope that they belong to multiple categories within a dimension.

Key Applications

When EAI is employed effectively, an enterprise can leverage its existing assets:

- To provide new products and services. An example is the provisioning of an online travel reservation application to employees so that by interacting with a single application, an employee can obtain approval for a business trip, book flights, and make reservations for hotel and car rental.
- To improve its relationships with customers, suppliers, and other stakeholders. An example is when an enterprise wants to obtain a consolidated view of all its business relationships with a customer.
- To streamline and improve its operations, for example, by reducing data redundancy and overlapping functions. An example is when mergers, acquisitions, or spin-offs happen. With these business activities come duplicated and fragmented operations such as multiple payroll systems, corporate directories, and so on, which can be streamlined.
- To simplify interactions among its applications by adopting a standard approach to integration. Once an integration strategy has been created and supported by an appropriate technical infrastructure, adding new participating applications or developing new applications with integration as a key consideration will no longer be as difficult.

Cross-references

- ▶ Business Process Execution Language
- ▶ CORBA
- ▶ Enterprise Service Bus
- ▶ Java Database Connectivity
- ▶ Messaging Systems
- ▶ Multi-Tier Architecture
- ▶ Open Database Connectivity
- ▶ RMI
- ▶ Service Component Architecture (SCA)
- ▶ Service Oriented Architecture
- ▶ Web Services
- ▶ XML

Recommended Reading

1. Hohpe G. and Woolf B. Enterprise Integration Patterns – Designing, Building, and Deploying Messaging Solutions. Addison Wesley, Reading, MA, 2004.
2. Adams J. IBM Patterns for e-business: application Integration pattern. IBM developerWorks. http://www-128.ibm.com/developerworks/patterns/application/index.html
3. Trowbridge D., Roxburgh U., Hohpe G., Manolescu D., and Nadhan E.G. Integration Patterns. Microsoft Corporation, 2004.
4. Ruh W., Maginnis F., and Brown W. Enterprise Application Integration – A Wiley Tech Brief. Wiley, New York, 2001.
5. Sauter G., Mathews B., Selvage M., and Lane E. Information service patterns, part 1: Data federation pattern. IBM developerWorks, July 28, 2006. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv1/
6. Sauter G., Mathews B., Selvage M., and Ostic E. Information service patterns, part 2: Data consolidation pattern. IBM developerWorks, Dec 5, 2006. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv2/
7. Sauter G., Mathews B., and Ostic E. Information service patterns part 3: Data cleansing pattern. IBM developerWorks, Apr 6, 2007. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv3/
8. Chari K. and Seshadri S. Demystifying integration. Commun. ACM, 47(7):59–63, 2004.
9. Dreibelbis A., Hechler E., Mathews B., Oberhofer M., and Sauter G. Information service patterns, Part 4: Master data management architecture patterns. IBM developerWorks,

Mar 29, 2007. http://www.ibm.com/developerworks/db2/library/techarticle/dm-0703sauter/
10. Gullege T. What is integration? Ind. Manage. & Data Syst., 106(1):5–20, 2006.

Enterprise Content Management

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms
ECM; Office automation; Records management; Document management

Definition
Enterprise content refers to the collections of records and documents that are used in support of business processes. Much of this data is unstructured or semi-structured text created by word processors and other productivity software as part of an enterprises' standard operating procedure. Enterprise content management applies database principles to collect, organize, index, and preserve such data, and ECM systems provide facilities for search, browsing, update, workflow management, web content management, collaboration support, version control, access control, record retention and destruction, legal compliance, and quality control.

Office automation typically refers to the components that create individual documents and manage the workflow. Records management refers to the components that protect and preserve the data and ensure that it is eventually archived or destroyed according to the enterprises' policies.

Key Points
Until recently, unstructured data was usually stored on the individual workstations of the office personnel who created them. This made the data inaccessible to those who wished to apply business intelligence procedures. Enterprise content management systems provide centralized mechanisms to access and manipulate the data, sometimes requiring that the data itself be centralized but often supporting distributed (federated) data management.

In essence, enterprise content management applies database principles and practices to the diverse forms of data that were traditionally outside the purview of relational database management systems.

Cross-references
- ▶ Access Control Administration Policies
- ▶ Document Databases
- ▶ Regulatory Compliance in Data Management
- ▶ Workflow Management

Recommended Reading
1. Jenkins T. Enterprise Content Management: What You Need to Know, Open Text Corporation, Waterloo, ON, Canada, 2004.

Enterprise Information Integration

- ▶ Information Integration

Enterprise Service Bus

Greg Flurry
IBM SOA Advanced Technology, Armonk, NY, USA

Synonyms
ESB; Service bus

Definition
The enterprise service bus (ESB) is a key architectural pattern in the larger architectural pattern called service-oriented architecture (SOA). The ESB supplies loosely coupled connectivity between service requesters and service providers in service-oriented solutions. Loose coupling permits a clean separation of concerns (temporal, technological, and organizational) between the parts in a solution, enabling flexibility and agility in both business processes and IT systems.

Historical Background
SOA is perhaps the latest stage in the continuing evolution of distributed computing. SOA inherits many principles form earlier stages of distributed computing. Client/server introduced the notion of less monolithic application architectures. Distributed objects focused on smaller-grained function and introduced encapsulated behavior and well-defined interfaces. Web services focused on the use of standards to enhance

interoperability. SOA brings a strong focus on the principles of service definition according to business value instead of IT value, service composition via business processes using choreography, and separation of concerns to promote reuse, flexibility and agility.

The ESB contributes greatly to the principle of separation of concerns in SOA. Similar to SOA, the ESB is perhaps just the latest stage in the continuing evolution of the connectivity used in distributed computing and inherits principles from earlier stages. Various messaging systems (sometimes called message-oriented middleware) enabled direct point-to-point interaction. Enterprise application integration introduced centralized management of indirect connectivity and the ability to adapt disparate applications or data sources. Web services emphasized the notion of well-defined interfaces, the use of standards and a service registry to facilitate dynamic behavior. The ESB, in the context of SOA, strengthens the focus on providing loosely-coupled, highly dynamic interactions between service requesters and service providers, whether they adhere to standards or not.

Foundations

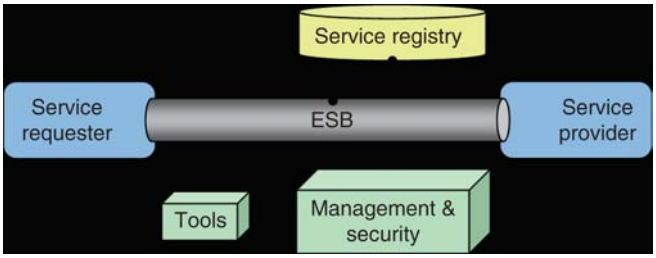
The architectural pattern called the ESB provides the connectivity that enables service interaction in SOA. As illustrated in Fig. 1, in any service interaction there are two roles, the service provider which offers a business function and the service requester that needs the business function. Service requesters and providers interact by exchanging messages. So more precisely, the ESB architectural pattern provides connectivity between service requesters and service providers. The ESB acts as a logical intermediary in the interactions; it intercepts messages from the requester, performs processing, often called mediation, on those messages, and sends the processed messages to the provider. Thus the ESB enables loosely-coupled service interactions. Loose coupling permits a clean separation of concerns (temporal, technological and organizational) between application services to enable flexibility and agility desired from SOA.

The ESB can be physically realized in different ways. The ESB appears like a centralized hub in Fig. 1, and the physical realization of the architectural pattern in many solutions in fact is a hub. The ESB, however, can be distributed so that mediation can physically take place in the service requester's environment, the service provider's environment, in one or more hub environments, or in any combination, allowing for optimization based on the requirements of the solution.

The ESB and Connectivity

Service requesters and service providers implement the application or business logic in a solution, targeted at achieving domain-specific business goals. The ESB implements the connectivity logic in a solution, targeted at achieving loosely-coupled, flexible, and on-demand interactions between service requesters and providers. In an ideal service oriented solution, separation of business logic and connectivity logic is clean, meaning that the services implement no connectivity logic, and that the ESB implements no business logic. Only by architecting this clean separation can an enterprise achieve the flexibility, agility and reuse sought from SOA.

It is sometimes difficult to distinguish between business logic and connectivity logic. There are guidelines, but the distinction may depend on the nature of the organization, or even a particular situation within the organization. One guideline leverages the distinction between semantics and syntax. A service provider performs actions which depend on the semantics, or meaning, of a service request, and so has to understand the meaning of the request; e.g., it creates, reads, updates or deletes business related



Enterprise Service Bus. Figure 1. Enterprise Service Bus.

entities. The ESB compensates for any mismatches in the syntax, or form, of the request between the service requester and provider, and has no need to understand the meaning of the request and does nothing to implement the semantics implied by the request. A related guideline emphasizes the distinction between achieving business objectives versus simply enabling interaction; since the ESB is an intermediary, removing the ESB should have no impact on logically achieving the business objectives; in other words, it adds no semantic value to a service interaction; put another way, the ESB never provides services, it always simply connects services.

Part of the difficulty distinguishing business logic and connectivity logic comes from the reality that the connectivity logic may itself be defined or driven by business considerations. For example, the ESB may route service requests to one of several candidate providers based on factors important to the proper functioning of the business, such as priority attributed to the requester or the time of day.

ESB Core Principles

Support for various forms of mediation allows the ESB to fulfill two core principles in support of separation of concerns. The first principle is service virtualization. The term refers to the ability of the ESB to compensate during service interactions for syntactic differences in:

- Protocol & pattern – Interacting services need not use the same communication protocol or interaction pattern. For example, a requester may require a single monolithic interaction via HTTP, but the provider may require interaction via some message-oriented protocol, using two correlated interactions. The ESB provides the conversion capability needed to mask the protocol and pattern switch.
- Interface – Service requesters and providers need not agree on the interface for an interaction. For example, the requester may use one form of message to retrieve customer information, and the provider may use another form. The ESB provides the transformation capability needed to reconcile the differences.
- Identity – A service requester need not know the identity (e.g., address) of the service provider, or vice versa. For example, service requesters need not be aware that a request could be serviced by any of several candidate providers at different physical locations. The actual provider is known only to, and is chosen by the ESB, and in fact, may change without impact to the requester. The ESB provides the routing capability needed to mask identity.

The second core principle is aspect-oriented connectivity. Service oriented solutions include multiple cross-cutting concerns like management and security. The ESB can implement or enforce such cross-cutting concerns on behalf of service requesters and providers, removing such concerns from of environment of the requesters and providers.

The application of these core principles allows the ESB to affect the qualities of service in interactions. Some aspects of an interaction can be abstracted away from the participants. Consider auditing, for example; if a solution requires auditing, the ESB can "add" it to interactions with no impact on the services. Similarly, the ESB can "add" or perhaps "enhance" qualities of service by retrying failed interactions, or in more sophisticated situations matching requester requirements with provider capabilities.

The ESB in an SOA Context

Figure 1 shows that the ESB performs its role in the larger context of SOA working with other parts of the architecture. The service registry (sometimes called a service repository or service registry/repository) contains and manages the metadata present in a service oriented solution; examples include interface descriptions, endpoint address and policies covering service level agreements, security relationships and so on. This service metadata contained in the registry, and thus the registry itself, have a broad scope in service oriented solutions, spanning governance, development and administration as well as runtime. The ESB can provide value using only static metadata provided during development, but realization of fully dynamic service virtualization and aspect-oriented connectivity requires the ESB to access a service registry at runtime to implement the dynamic connectivity required in a solution. Thus, the service registry can be considered the preferred way to configure the ESB, i.e., the service registry is a policy administration point for the ESB, and the ESB can be considered a policy enforcement point for the registry. As a result, the service registry, while not part of the ESB due to it broader role in SOA, is characterized as tightly coupled to the ESB.

Note that some important capabilities of any service oriented solution, specifically those related to management, are shown independent of, but accessible by the ESB. Management capabilities such as security and IT monitoring and management have a solution-wide scope, and require the coordination and cooperation of parts of a solution beyond the scope of the ESB. The ESB does not provide connectivity between services and management capabilities, and it is possible to secure and manage a solution without participation of the ESB. It is also possible, and frequently desirable, for the ESB to take an explicit, active role in helping secure and manage a solution as part of aspect-oriented connectivity. In this situation, the security and management policies are set by the policy administration points outside the ESB, and one can consider the ESB as a policy enforcement point and sometimes a policy decision point for such policies. Thus policy is set using management and security services that have no direct relationship to the ESB, but the ESB helps enforce the policy. As a result management capabilities are characterized as loosely coupled to the ESB.

An ESB requires tools for development and administration of the ESB. Developers use tools to develop the connectivity logic, or mediations, running in the ESB. Similarly, administrators use tools to deploy mediation and administer the ESB post-deployment. Ideally such tools leverage the service registry. For example, development tooling might allow the mediation developer to find the service providers required for an interaction using the registry; administration tooling might allow addition, deletion or modification of service metadata intended to impact the dynamic behavior of a solution.

Key Applications

An enterprise service bus provides the loose coupling necessary to allow service oriented applications to achieve the desired degree of separation of concerns.

Cross-references

- ▶ Service Component Architecture (SCA)
- ▶ SOA

Recommended Reading

1. An introduction to the IBM enterprise service bus. Available at: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-progmodel4/index.html. July, 2005
2. Discover how an ESB can help you meet the requirements for your SOA solution. Available at: http://www-128.ibm.com/developerworks/architecture/library/ar-esbpat1/. April, 2007
3. Enterprise service bus. Available at: http://en.wikipedia.org/wiki/Enterprise_service_bus. September, 2008
4. The enterprise service bus: making service-oriented architecture real. Available at: http://researchweb.watson.ibm.com/journal/sj/444/schmidt.html. 2005

Enterprise Terminology Services

Lawrence Gerstley
PSMI Consulting, San Francisco, CA, USA

Definition

Enterprise Terminology Services refers to the complete lifecycle of vocabulary creation, publication, and supporting processes for the entire electronic medical record system. The supporting processes may include quality assurance, search and retrieval supporting other systems, and mapping for interoperability. Supporting systems, such as managing workflow and regular publication, are also discussed.

Historical Background

One repeatedly encounters the same missing processes and system components required to enable a production level terminology system. In many cases, almost all of the focus is put onto the terminology and storage system, and terminology workflow, publication control, QA, and other sub-systems are added on as afterthoughts. The terminology system itself should have supporting structures, including historical data, metadata, publishing data, and workflow data to support a system that can be fully maintained and utilized by diverse groups.

Tooling in Terminology systems may be minimal or non-existent; existing enterprise-owned applications (such as an office suite of word processing, spreadsheets, etc.) are used to fill the voids. This often results in the author, editor, and user workflows being "shoe-horned" into the tools, rather than the tools being adapted to enable maximum data quality, efficiency, and clarity. The core terminology system and system tooling should be thought of together as the entire Enterprise Terminology System, and should evolve together to fit the needs of the users with the users' direct input.

Import and export of data to other internal enterprise systems, as well as importing and mapping to and from industry standards and vendors (such as SNOMED-CT, LOINC, ICD-9/10, CPT, First Databank/Medispan, etc.) needs to also be considered as a fundamental part of a complete system. Regular maintenance of supporting data, its impact on the enterprise terminology, and required quality assurance to approve changes can range from a few hours to months of effort. Planning and system tuning can pay back any planning and expenditures many times over.

Foundations

System Architecture

1. Core terminology database. Core data model itself may use established models, or be custom developed.
2. Supporting metadata to indicate individual row authorship, last changes, metadata commentary, Dublin Core, and other institution-specific information.
3. Human Workflow, providing complete terminology lifecycle tracking for terminology requests, overall system requests, ad hoc projects, and other project tasks that require tracking.
4. QA/Metrics Reporting layer, providing insights into overall current status, historical throughput, publishing status, and other system metrics.
5. Data interchange layer, providing services for import and export of data to the system, ideally supporting HL7 messaging as a subset of interchange.
6. Centralized knowledgebase for sharing of information throughout the authoring group and end-users.

Key Applications

Case Studies and Experiences

Experience with multiple organizations shows a surprising overlap in system issues and organizational challenges, despite different company work styles, authoring teams, and end-users. In most cases, the problems were due to expediency of getting a working system in place, a philosophy of "add it later" when missing systems were suggested during design phases or identified in the normal course of work. One interesting note is that in a majority of cases, users and managers turn immediately to a spreadsheet application to fill system functionality gaps, often with disastrous consequences.

Users employ spreadsheets to manage small projects, stage data before loading into the authoring system, and other ad hoc tasks. While spreadsheets provide wide flexibility in usage, their use often has unexpected side effects. For example, groups using spreadsheet exhibit internal codes and foreign keys changing through the dropping of leading and trailing zeroes and rounding of decimal places. Intended to aid in the creation of financial information, which was the foundation of spreadsheets, these transformations have serious effects to ICD-9 codes, foreign key mapping, and other scenarios where keys must be treated as immutable. When such functional deficiencies are discovered by authoring groups, attempts are often made to make the spreadsheet application more rigid by setting software options to be standard in the group (disabling automatic formatting), developing internal templates that users should not deviate from, and potentially even developing programmatic control (such as scripting languages like VBA) behind the spreadsheet's GUI.

This is a natural outcome of finding procedural weaknesses; however, the investment of effort is wrongly placed, creating an organic system that is created one piece and a time, creating a fragile and overly complex system. The problem stems from not using the correct technology for the task. If the task is to stage data for loading into the system, experience shows that the creation of a flexible staging model and authoring tools to create terminology in the same system that will ultimately store the verified and validated data centralizes the information so multiple individuals may collaborate. Additionally, the actual tasks of verification and validation are performed instantly making the data available in the production environment.

When spreadsheets are used to perform project management, problems arise when multiple versions of spreadsheets are distributed, updated by multiple individuals, passed in emails, and propagated through a variety of other means, confusion as to which spreadsheet acts as the source of "truth" reigns. Again, this results from employing the wrong tool for the task. Groups often employ project management tools, but such tools focus on the project manager, resulting in a single user overseeing a set of tasks. Terminology authoring groups often need to see overall progress of the active project. Additionally, work may need to be performed piecemeal by team members who can self-assign individual items to themselves from a centralized pool of outstanding work. These challenges are ideally

suited to a human workflow engine, such as Serena Software's TeamTrack or K2's workflow products. Such systems provide the ability to create centralized, automated workflow entry, tracking, and reporting with workflows that can be modeled to accurately represent the way that group's work. Additionally, using such systems focuses the group's attention on methods of improving workflow, gaining efficiencies and accuracy, obtaining approvals, and recording those approvals from requesters and other interested parties.

In general, turning to open-source software products to help fill in the process gaps. Firstly, it permits selection of a software tool actually geared towards the problem in question. Too often software is chosen because it is already part of a group's inventory. As with spreadsheets, its use *can* be adapted to solve the problem at hand, but results in changing the group's workflow, de-scoping the desired functionality, and other short cuts. The effects of these ostensibly expedient decisions will be encountered later, and as noted earlier will lead to subsequent additional processes, short cuts, and errors. Using open-source not only allows finding close match between requirements and desired functionality, but also permits tight integration between systems by altering or extending the code base. Such a solution is often more palatable to IT and other engineering groups, and begins to develop a completely integrated work environment. The normal purchasing process is removed from the start-up time, and approvals usually become only technical approvals.

Standardizing and expressing editorial policies formally in a computationally leverageable manner yields many benefits. Using a very simple example, expressing the standards for term string display may include capitalization rules, maximum string length for a particular string, and spaces after different punctuation marks can be expressed as part of a centralized system parameter file. Using an XML format to express such rules delivers many benefits:

1. Such a file is human-readable, providing more transparency into the systems operation.
2. It is technology-agnostic, which means that any future application can process the file for use through standard libraries, regardless of language or platform.
3. Changes may be made easily through any number of tools, ranging from the simplest of text editors (potentially dangerous) to XML editors.
4. As an XML file, it may be versioned with any in-house versioning tool already employed for software control (e.g., CVS, Subversion, SourceSafe, ClearCase).
5. Through open-source technologies like XSLT and FO, the file can be self-documenting, generating formal PDF, Word, RTF, and other test formats for system documentation.

Terminology data models often serve as the focus of development, but often this model is developed to represent the universe of active terminology in a production-ready state. Consequently, it may have little in it supporting developing terminology that must go through a verification and validation process before it is released to outside users. Metadata supporting authoring, publication control, and quality assurance is often missing from the data model, which creates the need for supplemental systems to represent and alter such information.

To illustrate, consider the need to create a new diagnosis in the system, which must be mapped to the appropriate ICD-9/10 code(s) for that diagnosis. Until such a time as a certified coder has looked at the display string, potential synonyms, and the code itself, and certified that code as accurate, such a diagnosis should not be published for outside use. However, putting the data into the authoring system itself means that either the publication process knows not to include the record until it has been verified, or the entire publication of updated terminology must be held until all records are in a satisfactory, verified state (the "all-or-nothing" method of publishing). Sometimes, groups get around this by staging data outside of the authoring system until it is ready for loading, but this brings about additional issues, including all of the previously mentioned issues surrounding use of spreadsheets and other non-authoring applications. Additionally, staging data may lead to a situation where multiple updates to the same record may be prepared outside of the system, and such work will overwrite one another. This scenario often arises during coding updates, or quality assurance and auditing tasks.

These issues necessitate an authoring model that can support records in various stages of completion. Also, it should support different levels of record locking, so that others may see a record in a "dirty" status and be able (or unable) to alter it depending upon their system role. Such a model should include

metadata about the terminology records themselves, including:

1. Which user created (initiated) the record?
2. When the record was created?
3. Which user last updated the record?
4. When the last update occured?
5. Commentary about the row itself.
6. Record status (dirty, draft, awaiting publication, propagated).
7. Record publication date/time.

Once an effective set of controlling metadata is in place tracking terminology flow and validation, maintenance routines may begin updating coding and other integrated data, such as pharmaceutical information. Too often such maintenance routines grow without any control over languages used, processes requiring automation, or even documentation for completing updates and handling diverse use-cases. This is the result of different individuals creating processes as time permits, with the tools that they are already familiar with, accomplishing what needs to be done at that point in time. The same maintenance process may contain a potpourri of technologies, each employed at a different point in the process. In general, a small set of tools should be selected and standardized for maintenance, and full documentation should be an absolute requirement for system maintenance. For scripting languages, Perl, Python, and Ruby will all do the job and usually fit within any group's existing IT environment.

Defined metrics reporting should be designed on top of the terminology model, and accessible to those within the team as well as management. Effective reporting systems have been assembled from purchased packages, such as Crystal Reports, as well as open-source packages, such as BIRT, PHP and Apache Cocoon. Cocoon is intriguing in that it uses data "pipelines" which defines how the data is created, transformed, and finally delivered to the requester. Pipelines work with XML structures, so the output at any stage can be brought into other systems. Cocoon also provides a separation between the generation and presentation of data, so that the same pipeline can generate PDF, XML, Doc, HTML, and a number of other formats.

Terminology systems are often the result of evolutionary development, and can benefit greatly from redesign and standardization. Although groups often balk at the time or investment that such a redesign would take, often improvement in just one of the aforementioned areas will pay back such an investment several fold. Additionally, the system will have improvements of all other areas. Internal standardization of the terminology system also prepares it for eventual integration with outside systems, which is an industry goal.

Cross-references

- ▶ Business Intelligence
- ▶ Business Process Management
- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Controlled Vocabularies
- ▶ Data Integration
- ▶ Data Transformation
- ▶ Database Management System
- ▶ Dublin Core
- ▶ Information Lifecycle Management
- ▶ Meta Data
- ▶ Pipeline
- ▶ Process Optimization
- ▶ Process
- ▶ Storage Management
- ▶ XML Information Integration
- ▶ XML
- ▶ XSL/XSLT

Recommended Reading

1. Apache cocoon. Available at: http://cocoon.apache.org
2. Docbook. Available at: http://www.docbook.org
3. Dublin core metadata initiative. Available at: http://dublincore.org/
4. Fogel K. and Bar M. *I*CVS: Open Source Development with CVS, 3rd edn. Paraglyph, Scottsdale, AZ, 2003.
5. Leymann F. and Roller D. Production Workflow, Concepts and Techniques. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
6. Walsh N. and Muellner L. DocBook: The Definitive Guide. O'Reilly, Sebastopol, CA, 1999.

Entity Relationship Model

Il-Yeol Song[1], Peter P. Chen[2]
[1]Drexel University, Philadelphia, PA, USA
[2]Louisiana State University, Baton Rouge, LA, USA

Synonyms

ER Model; ERM; Entity-Relationship Model

Definition

The entity relationship model (ERM) is a conceptual model that represents the information structure of a problem domain in terms of entities and relationships. The result of modeling using the ERM is graphically represented as an entity relationship diagram (ERD). Thus, an ERD represents the conceptual structure of a problem domain being modeled. ERDs are widely used in database design and systems analysis to capture requirements of a system or a problem domain. In particular, when used for data modeling, the ERD assists the database designer in identifying both the data and the rules that are represented and used in a database. ERDs are readily translated into relational database schemas.

Historical Background

The ERM was introduced by Peter Chen in 1976 [2]. The ERM and its variations have been widely used in database modeling and design, systems analysis and design methodologies, computer-aided software engineering (CASE) tools, and repository systems. The popularity of the ER approach led to many extended ER and semantic data models as well as to its variations in notations of ERDs [9]. The ER model and its related research work have also laid the foundation for object-oriented analysis and design methodologies, which led to the development of the Unified Modeling Language (UML). Chen [4] presents historical events related to the ERM up to the early 2000s and roles of the ERM on the development of CASE tools.

Foundations

The three basic modeling components of the ERM are entities, relationships, and attributes.

Entities, Entity Types, and Attributes

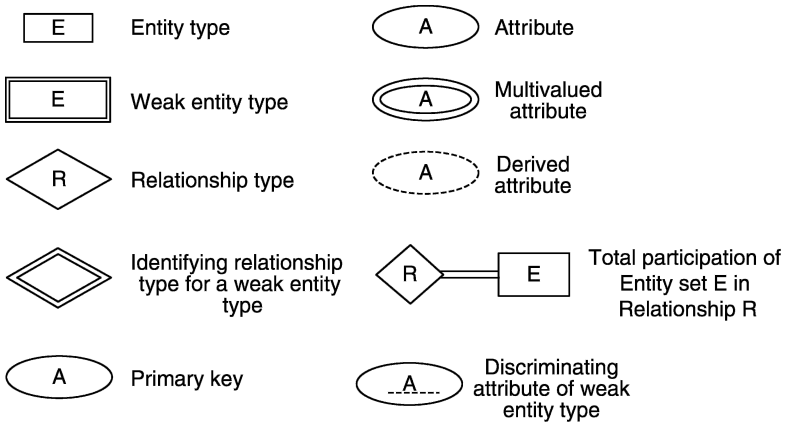
An *entity* is a primary object of a problem domain about which users need to capture information. An entity or an *entity instance* (or *entity occurrence*) is an object in the real world with an independent existence. An entity is characterized by its properties called *attributes*. Thus, an entity can be differentiated from other objects when it has at least one property different from the others. An *entity type E* defines a collection of entities that have the same attributes. For example, "John Smith" can be an entity instance of the entity type called *Employee*. Attributes of the *Employee* entity type could include *first name*, *last name*, *social security number*, *date of birth*, *address*, and *salary*. Attributes of a *Customer* entity type could include *customer number*, *first name*, *last name*, *address*, and *customer rating*. Thus, the *Employee* entity type is distinguished from the *Customer* entity type because they have different attributes. In the original ERM, all the attributes are assumed to be simple atomic values, representing numbers, character strings, and dates, not multi-valued attributes such as sets or structures.

Typical entity types usually belong to certain entity categories such as roles of people, locations, tangible things with values, organizations, events, transactions, etc. Thus, *Employee*, *Store*, *Product*, *Company*, *Reservation*, and *Sale* are all examples of entity types. In the literature, an entity and its entity type are not frequently distinguished from each other because the distinction between them is usually clear in the context.

An entity type has two types of properties: identifying attributes and descriptive attributes. An identifying attribute uniquely determines each instance of an entity type. An identifying attribute is called an entity identifier or a *key attribute*. For every entity type, there must be a designated key. For example, the attribute *social security number* would uniquely identify each instance of the entity type *Employee*. Hence, *social security number* is called the entity identifier or the primary key of the *Employee* entity type. A descriptive attribute describes a non-unique property of an entity type. Descriptive attributes of the *Employee* could include *first name*, *last name*, *date of birth*, *address*, and *salary*. Only those attributes that are meaningful in the problem under consideration are included in the ERD. For example, *eye color* would not be included in the *Employee* entity unless one needed to use *eye color* in a meaningful way, e.g., using it in an eye-color activated security system.

An attribute of an entity instance has a *value*. For example, "John" is one value of the attribute *first name*. The *domain* of an attribute is the collection of all possible values an attribute can have. For example, the domain of *first name* is a character string and that of *salary* is a number.

Figure 1 summarizes the notation used in ERDs. An entity type is depicted as a *rectangle* containing the name of the entity type. The name of an attribute is enclosed in an *oval* connected to the rectangle of the entity type they describe. An entity identifier or a key is underlined.



Entity Relationship Model. Figure 1. Symbols used in an Entity Relationship Diagram. (The notation for total participation was adopted from Elmasri and Navathe [6]).

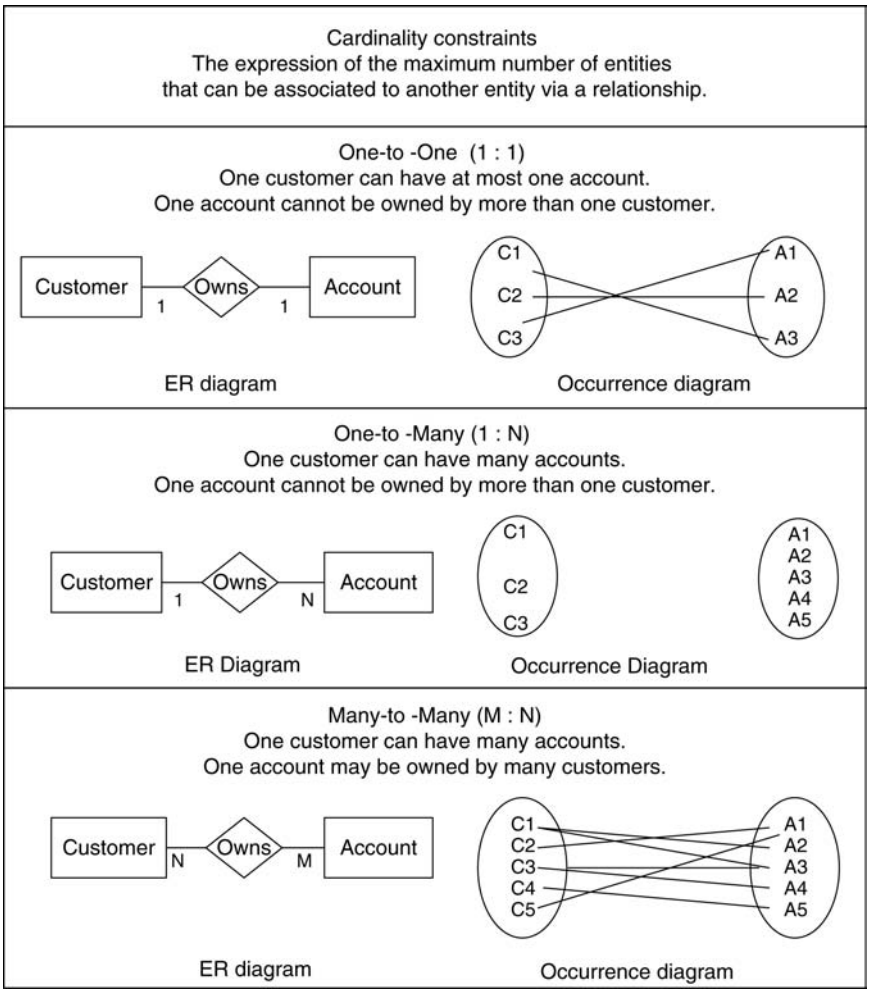
Relationships and Relationship Types

A relationship is an association between or among entities. A relationship in the ERM describes a meaningful association that needs to be remembered between or among entities. A relationship type between entity types E and F meaningfully relates some entities in E to some entities in F. Formally, a *relationship* R is a Cartesian product of $n \geq 2$ entities$\{R\ (e_1, e_2,...,e_n) \mid e_1 \in E_1, e_2 \in E_2,...,e_n \in E_n\}$, where e_i is an entity instance, E_i is an entity type, and $R\ (e_1, e_2,...,e_n)$ is a relationship. Here, n is called the *degree* of a relationship R, indicating how many entity types are participating in R. When $n = 2$, R is called a binary relationship; and when $n = 3$, R is called a ternary relationship. That is, a binary relationship is a meaningful association between two entity types, and a ternary relationship is a meaningful association among three entity types. In typical modeling, binary relationships are the most common and relationships with $n > 3$ are very rare. In a special binary relationship in which $E1 = E2$, the relationship is called a recursive relationship because an entity is related to another entity of the same type. That is, a recursive relationship is a meaningful association between entity instances of the same entity type. As shown in Fig. 1, relationships are depicted by *diamonds* between or among entities in the ERD.

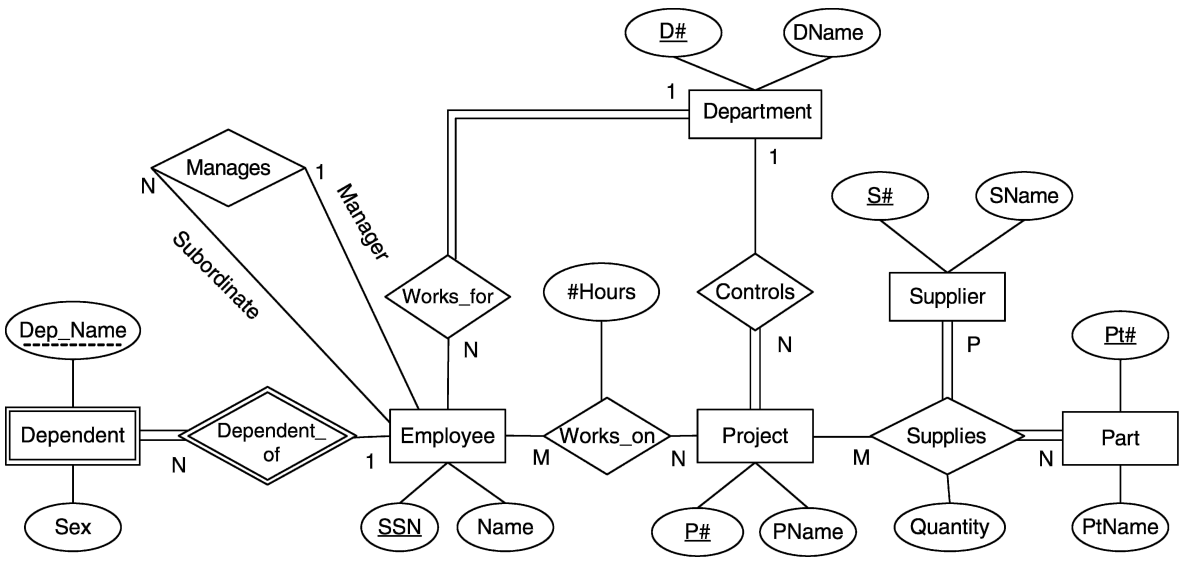
Cardinality and Participation Constraints

Cardinality is a constraint on a relationship between two entities. Specifically, the cardinality constraint expresses the maximum number of entities that can be associated with another entity via a relationship type. The values of cardinality are either "one" or "many". They are usually represented by either "1" or "N" in an ERD. A binary relationship can have three possible cardinalities: one-to-one (1:1), one-to-many (1:N), or many-to-many (M:N). Figure 2 shows the three cardinality constraints between two entity types *Customer* and *Account*. One-to-one cardinality states that one *customer* can have at most one *account* and one *account* can be owned by at most one *customer*. One-to-many cardinality says that one *customer* can have many *accounts*, but one *account* cannot be owned by more than one *customer*. Many-to-many cardinality says that one *customer* can have many *accounts* and one *account* may be owned by many *customers*. Figure 3 shows examples of a recursive relationship, binary relationships, and a ternary relationship as well as various cardinality constraints.

Participation is also a relationship constraint. Participation expresses the minimum number of entities that can be associated with another entity via a relationship type. There are two values for participation: *total* (also known as *mandatory*) participation and *partial* (also known as *optional*) participation. If every instance of an entity must participate in a given relationship, then that entity has total participation in the relationship. But if every instance need not participate in a given relationship, then the participation of that entity in the relationship is partial. In an ERD, the total participation of Entity type E to relationship type R is indicated by a double line from E to R [6], as shown in Fig. 1. For example, in Fig. 3, the participation of the *Department* entity type in the *Works_for* relationship is total, while that of the *Employee* entity type in the relationship is partial. This implies that a *Department* instance must



Entity Relationship Model. Figure 2. Connectivity of binary relationships.



Entity Relationship Model. Figure 3. An example ERD with different types of relationships.

have at least one *Employee,* while an *Employee* instance can exist without working for a *Department.*

Cardinality and participation constraints are business rules in the problem domain being modeled. They represent the way one entity type is associated with another entity type. They are also integrity constraints because they help ensure the accuracy of the database. These constraints limit the ways in which data from different parts of the database can be associated. For example, say the cardinality of the relationship between *Customer* and *Account* is one-to-one, as in Fig. 2. If customer C1 is associated with account A3, then C1 cannot be associated with any other accounts and A3 cannot be associated with any other customers.

Attributes of a Relationship

A relationship sometimes has attributes. A relationship attribute represents a property of the relationship, but not of any participating entity type. For example, in Fig. 3, attribute *#Hours* is a property of the relationship *Works-on*, not of *Employee* or *Project.* Another example is the *Quantity* attribute of the ternary relationship *Supplies* in Fig. 3, where *Quantity* is a property created by the interaction of three participating entity types *Supplier*, *Project*, and *Part.* Relationship attributes are most common in many-to-many relationships or ternary relationships.

Roles

The meaning of a relationship type is usually clear between two associated entity types. However, the meaning of a relationship type is not clear if there are multiple relationship types between the participating entity types. In these cases, roles are used to indicate the meaning of an entity to the associated relationship type. Roles are indicated in ERDs by labeling the relationship lines that connect diamonds to rectangles. For example, a recursive relationship always has two lines between an entity type and the recursive relationship. In Fig. 3, *Manages* is a recursive relationship; *Manager* and *Subordinate* are roles of *Employee* entity instances. Thus, the relationship can be read as "an employee with the role of a manager can manage many subordinate employees, and an employee with the role of subordinate can be managed by one manager employee." Thus, in recursive relationships, it is customary to add roles to clarify the meaning of each relationship line. In other occasions, role labels are optional.

Ternary Relationships

A ternary relationship is a meaningful association among three entity types. An important requirement of a ternary relationship is that the three entity types must always occur at the same time in the relationship. For example, in Fig. 3, *Supplies* is a ternary relationship among the *Supplier*, *Project*, and *Part* entity types. The semantics of *Supplies* can be read as follows [7,12]:

- For a given pair of Supplier and Project instance, there are many Part instances.
- For a given pair of Part and Supplier instance, there are many Project instances.
- For a given pair of Part and Project instance, there are many Supplier instances.

Weak Entity Types

An entity type that does not have its own unique identifier is referred to as a weak entity type. A weak entity type has one or more *owner* (or *strong*) entity types connected through one or more one-to-many relationships (called *weak relationships*). Therefore, the primary key of a weak entity type is always composite. The key consists of the primary keys of all the owner entity types and a *discriminator (or partial key)* of a weak entity set, where the discriminator is the set of attributes that distinguishes among all the entities of a weak entity.

As shown in Fig. 1, a weak entity type is depicted by double rectangles, a weak relationship by double diamonds, and a discriminator is underlined with a dashed line. For example, in Fig. 3, *Dependent* is a weak entity, *Dependent_of* is a weak relationship, *Employee* is the owner entity type of *Dependent*, and *Dep_Name* is the discriminator of *Dependent.* Thus, the primary key of *Dependent* is a composite of *Employee.SSN* and *Dependent.Dep_Name.*

Two important properties of a weak entity type are the *identifier dependency* (ID-dependency) and the *existence dependency.* The ID-dependency means that the primary key of the owner entity type is included in that of the weak entity type. Therefore, a weak entity always has total participation in its weak relationship, as shown in Fig. 3. Due to this property, the weak relationship is also called the identifying relationship. The existence dependency means the existence of a weak entity instance is dependent on the existence of its related owner entity instance. For example, in Fig. 3, if an *Employee* entity instance is deleted, then

all of its associated *Dependent* entity instances must also be deleted.

ER Modeling Techniques

Developing a syntactically and semantically correct ER model for a domain requires both effective modeling techniques and experience. Techniques for developing a correct and complete ERD are beyond the scope of this article. An excellent starting point for understanding techniques of constructing ERDs from a problem statement is presented by Chen [3]. Chen presents a fundamental framework for developing an ERD from English sentence structure by showing the similarity between English grammar and ER modeling components. Batini et al. [1] present conceptual modeling using ERMs. Song et al. [9] present a comparison of popular ERD notations as well as a comparison between *n*-ary models and binary models. Ling [8] presents a notion of a normalized ER diagrams. Dullea et al. [5] discuss validity and redundancy in ER diagrams. Teorey et al. [10] cover in detail the database modeling and design using the ERM.

Translation of ERDs into a Relational Schema

This section briefly presents how to translate ERDs into relational schemas. There are several different methods for creating relational schemas from an ERD. See [6,10,11] for more detailed treatments of the subject. The one described here is the most popular technique [6,10]. The rules are:

1. Every entity type becomes a table.
   a. All the attributes of the entity become the attributes of the table.
2. Each 1:N relationship type is mapped into the associated N-side entity type as follows:
   a. For each N-side entity type:
      (a) Add the primary key (PK) of the 1-side entity type. This added PK becomes a foreign key.
      (b) Also add any descriptive attributes of the relationship.
   b. When this rule is applied to recursive relationships:
      (a) The PK of the 1-side entity type is annotated with its role name.
3. Each M:N relationship type becomes a separate table.
   a. Add PKs of the participating entity types to the table. They become foreign keys.
   b. Add all the descriptive attributes of the relationship type to the table.
   c. The PK of the table consists of the PKs of the two participating entity types.
   d. When this rule is applied to M:N recursive relationship type:
      (a) The two foreign keys are annotated with their role names.
4. Each 1:1 relationship type can be combined with either side of the entity type or can be treated like a 1:N relationship type.
5. A weak entity type becomes a table that includes the primary key of the identifying owner entity type.
   a. The PK of the table consists of the foreign key and the discriminator.
6. Each relationship type becomes a separate table.
   a. Add PKs of the participating entity types to the table. They become foreign keys.
   b. Add all the descriptive attributes of the relationship type to the table.
   c. The PK of the table consists of the foreign keys of the three participating entity types when the cardinality is many-many-many. When the cardinality is not many-many-many, the PK of the table consists of at least two foreign keys, where all the foreign keys coming from many-side entity types must be included in the PK of the table.

Applying the above translation rules to the ERD shown in Fig. 3 yields the following relational schema:

1. Employee (SSN, Name, D#, Manager_SSN).
2. Dependent (SSN, Dep_Name, Sex).
3. Department (D#, DName).
4. Project (P#, PName, D#).
5. Works_on (SSN, P#, #Hours).
6. Supplier (S#, SName).
7. Part (Pt#, PtName).
8. Supplies (P#, S#, Pt#, Quantity).

Key Applications

The ER model is widely used in database modeling and design and in conceptual modeling of information systems. Most commercial CASE tools support an ERM or some of its variations. The ER approach has been applied to design of object-oriented databases, data warehouses, temporal databases, spatial databases, and meta models. The ER model also laid the

foundation for class modeling techniques of object-oriented analysis and design.

URL to Code

An international conference that focuses on entity-relationship approaches and conceptual modeling has been held regularly since 1979. The conference is now known as International Conferences on Conceptual Modeling and is cataloged in http://conceptualmodeling.org [7].

Cross-references

- ▶ Conceptual Schema Design
- ▶ Database Design
- ▶ Extended Entity-Relationship Model
- ▶ Logical Database Design: from Conceptual to Logical Schema
- ▶ Meta Data Repository
- ▶ Object Data Models
- ▶ Semantic Data Model
- ▶ Temporal Logical Models
- ▶ Unified Modeling Language

Recommended Reading

1. Batini C., Ceri S., and Navathe S.B. Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings, Reading, MA, 1991.
2. Chen P.P. The entity relationship model – toward a unified view of data. ACM Trans. Database Sys., 1(1):9–36, 1976.
3. Chen P.P. English sentence structure and entity relationship diagrams. Inf. Sci., 29(2–3):127–149, 1983.
4. Chen P.P. Entity-relationship modeling: historical events, future trends, and lessons learned. In Software Pioneers: Contributions to Software Engineering, M. Broy, E. Denert (eds.). Springer, NY, 2002, pp. 100–114.
5. Dullea J., Song I.-Y., and Lamprou I. An analysis of structural validity in entity-relationship modeling. Data Knowl. Eng., 47 (3):167–205, 2003.
6. Elmasri R. and Navathe S. Fundamentals of Database Systems, 5th edn. Benjamin/Cummings, Reading, MA, (2007).
7. ER conferences. Available at: http://conceptualmodeling.org
8. Ling T.W. A normal form for entity-relationship diagrams. In Proc. 4th Int. Conf. on Entity-Relationship Approach. IEEE Computer Society, WA, 1985, pp. 24–35.
9. Song I.-Y., Evans M., and Park E.K. A comparative analysis of entity-relationship diagrams. J. Comput. Softw. Eng., 3(4): 427–459, 1995.
10. Teorey T.J., Lightstone S.S., and Nadeau T. Database Modeling & Design: Logical Design, 4th edn. Morgan Kauffman, San Francisco, CA, 2005.
11. Teorey T.J., Yang D., and Fry J.P. A logical design methodology for relational databases using the extended entity-relationship model. Comput. Surv., 18(12):197–222, 1986.

Entity Resolution

- ▶ Record Matching

Entity-Relationship Model

- ▶ Entity Relationship Model

EPN

- ▶ Event Processing Network

Equality Query

- ▶ Membership Query

Equality Selection

- ▶ Membership Query

Equality-Generating Dependencies

Ronald Fagin
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

egd

Definition

Equality-generating dependencies, or *egds*, are one of the two major types of *database dependencies* (the other major type consists of *tuple-generating dependencies*, or *tgds*).

To define egds, it is necessary to begin with the notion of an *atomic formula*, where an example is the formula $P(x_1,...,x_k)$, where P is a k-ary relational symbol, and $x_1,...,x_k$ are variables, not necessarily distinct.

Then egds are formulas of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow (x_1 = x_2))$, where

1. $\varphi(\mathbf{x})$ is a conjunction of atomic formulas, all with variables among the variables in $\mathbf{x}$.
2. Every variable in $\mathbf{x}$ appears in $\varphi(\mathbf{x})$.
3. x_1 and x_2 are distinct variables in $\mathbf{x}$.

Conditions (1) and (2) together are sometimes replaced by the weaker condition that $\varphi(\mathbf{x})$ be an arbitrary first-order formula with free variables exactly those in $\mathbf{x}$.

Key Points

The most important example of an egd is a *functional dependency*, where an example is the formula

$$\forall x_1 \ldots \forall x_k \forall x_i' (P(x_1,\ldots,x_k) \wedge P(x_1,\ldots,x_{i-1},x_i',x_{i+1},\ldots,x_k) \rightarrow (x_i = x_i')),$$

where P is a k-ary relation symbol.

Historically, functional dependencies were the first database dependencies. They were introduced by Codd [2] for the purpose of database normalization and design. Fagin [3] defined the class of *embedded implicational dependencies*, which includes both tgds and egds, but he focused on the case where they are (i) unirelational (so that all atomic formulas involve the same relation symbol) and (ii) typed (so that no variable can appear in both the ith and jth position of an atomic formula if $i \neq j$). Beeri and Vardi [1] defined and named tgds and egds.

Cross-references

- ▶ Database Dependencies
- ▶ Functional Dependency
- ▶ Normal forms and Normalization
- ▶ Tuple-Generating Dependencies

Recommended Reading

1. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
2. Codd E.F. Further normalization of the data base relational model. Courant Computer Science Series 6: Database Systems. Prentice-Hall, USA, 1972, pp. 33–64.
3. Fagin R. Horn clauses and database dependencies. J. ACM, 29(4):952–985, 1982.

ER Model

▶ Entity Relationship Model

ERM

▶ Entity Relationship Model

ESB

▶ Enterprise Service Bus

Escrow Transactions

Patrick O'Neil
University of Massachusetts, Boston, MA, USA

Synonyms

Escrow transactions

Definition

Escrow transactions [3] permit non-blocking concurrency for the most common high volume transactions, those that perform updates of hotspot data only by incrementing and decrementing aggregate data items. An example of such a transaction is one that increments dollar sale total of a customer for each purchase that decrements the quantity on hand of a product. A *hotspot* of a database under a high volume transactional workload consists of the set of data items, each one of which frequently needs to be updated by multiple concurrent transactions in order to maintain needed throughput. While this is impossible using read-write locking, since write operations do not commute, high-volume OLTP applications of this kind can be performed without blocking by a transactional system that performs increments and decrements with the appropriate protocols. Increment-decrement updates have been used for years in the IMS Fast Path product [1,2], supporting commuting high volume short-term transactions on data items in a centralized DBMS. Escrow transactions generalized this approach to support commuting long-lived transactions performing increment-decrement updates of aggregate data items, possibly in a distributed database.

Key Points

The term "escrow" is from banking: a large company entering into a business transaction in a remote

location would not want to "lock" it's main bank account for this purpose, but would elect an *escrow agent*, to hold the money in trust pending fulfillment of the transaction; if the transaction succeeds, the escrow agent would transfer the money, but otherwise the money would be returned to the company. Note that this is a long-lived transaction in the sense that a business transaction using escrow might take days or weeks to complete. In the same way the Escrow transactions perform quick increments and decrements to an account, then hold no locks on what quantity remains, and so lock only the portion of an aggregate quantity needed for the business purpose of the transaction.

Escrow transactions can be illustrated with an example of a product table *Prod* with a primary key *prid* and rows containing data about warehouse storage of a product, e.g., storage_capacity, price, etc. Updates to columns of Prod are infrequent and take part in short-term update transactions, but orders changing Quantity on Hand of the product are held long-term in an Escrow data item qoh, as explained below. The qoh item can be thought of as a *composite* column for each row of the Prod table, having the appropriate prid and a number of int fields, as follows:

(qoh, prid, val, inf, sup)

The qoh item must be exempted from standard transactional recovery using before images and after images, since Escrow recover is special-purpose as described below. Note that an increment operation Inc(qoh, prid, tid, delta) by transaction tid can represent either an increment (with a positive value of delta) or a decrement (with a negative value of delta). It is assumed that product orders such as this can be held pending for an extended length of time (weeks), and either committed or aborted at the end of that time, with a fee paid by the customer for this flexibility; thus each Inc operation can be thought of as an *Option to Buy (Return)* if delta is negative (positive).

The field val represents the value of the balance if all outstanding transactions that have incremented this quantity should abort, while the field inf(sup) represent the lowest (highest) values val could attain if all transactions that have performed Inc operations with positive-valued (negative-valued) delta increments were to abort, while all increments with deltas of the opposite sign commit. Thus as new Inc operations are submitted, negative-valued deltas will lower the inf value and positive valued deltas will increase the sup value, while val will remain unchanged until one of the transactions commits. In addition, as each Inc operation causes qoh items to change, an Escrow log (Elog) is created and both the Elog and the qoh are made durable (written to mirrored disk for example), to guarantee that the Elog will be recovered in event of a system crash. Recovery from a system crash will therefore result in recovery of qoh and all Elogs, so each Escrow update is brought back to a state where it can be either committed or aborted. There are also Max and Min field values in the descriptor of the qoh data item to provide a constraint on the maximum and minimum attainable values for val. The seller will disallow an Inc(delta) with negative delta which brings inf below Min (usually zero since this would mean that some purchase order will fail if no items are returned, while there will also be a limit on how many positive delta increments can be accepted for a given product (sup will be constrained to be less than Max), since there is a limited amount of storage. Back-orders for purchase or return can be provided when circumstances permit.

Note that while Inc operations commute, they do not commute with Reads or Writes. One cannot Read or Write an Escrow item that has indeterminate value (between inf and sup), although a Read function call to return this range of possible values could be provided.

Escrow Commits and Aborts

The commutative property of Escrow transactions is based on the fact that updates by multiple concurrent transactions on any single Escrow data item can be committed or aborted in any order. This is motivated with a short example. Assume the qoh data item on a product with prid = 1234 begins with val = 1000, and that Max = 2000 and Min = 0. A sequence of transactions T_i updates this qoh creating Escrow logs, and then all T_i Commit (C_i) or Abort (A_i).

To start: (qoh, prid: 1234, val: 1000, inf: 1000, sup:1000)
Inc_1: Inc(qoh, prid: 1234, tid: 1, delta: −500)
Result: (qoh, prid: 1234, val: 1000, inf: 500, sup:1000), Elog: (tid: 1, delta:−500, qoh[1234])
Inc_2: Inc(qoh, prid: 1234, tid: 2, delta: + 500)

Result: (qoh, prid: 1234, val: 1000, inf: 500, sup:1500), Elog: (tid: 2, delta: + 500, qoh[1234])
C_2: (qoh, prid: 1234, val: 1500, inf: 1000, sup:1500), Elog for tid = 2 was used and is now invalidated.
A_1: (qoh, prid: 1234, val: 1500, inf: 1500, sup:1000), Elog for tid = 1 was used and is now invalidated.

Recall that a positive increment such as Inc_2: Inc (qoh, prid: 1234, tid: 2, delta: + 500) causes the sup value to increase by delta. Then an Abort A_2, reading off the Escrow log for Inc_2, causes the sup value to be decremented by delta again, with no other field changed, while a Commit C_2 causes the inf, and val values to be incremented by the same amount (if no other transactions remain active, the val, inf and sup must end with the same values). Similarly a negative increment Inc_1: Inc(qoh, prid: 1234, tid: 1, delta: −500) causes the inf value to be decremented by the negative delta; an Abort A_2, reading off the Escrow log for Inc_1, will cause the inf value to be incremented by delta (a positive value), while a Commit will cause the val and sup values to be decremented as well.

It should be clear from this that the minimum value possible for val at the end of a sequence of Commits and Aborts will be inf, and the maximum value will be sup. Any combination of a specific set of Aborts and Commits in any order will result in the same val value.

Distributed Escrow Transactions

A classical definition of distributed transactions has a Transactional Coordinator on one host performing updates on its own host and coordinating updates on several foreign hosts to provide ACID properties for a global transaction. The updates lock records on all hosts in the classical situation, often leading to difficulties in terms of blocking, especially if one of the non-coordinating hosts should crash at an inopportune moment during concluding Two-Phase Commit. Escrow transactions are perfect for situations like this, since they lock only the (usually small) portion of an aggregate quantity needed for the business purpose of the transaction, while original Escrow update automatically requests Phase 1 of a Two-Phase Commit, and indeed even if the Coordinator host crashes after the final commit has started, one can afford to wait for recovery without blocking all the data items involved against further Escrow updates.

Cross-references

- ▶ Concurrency Control
- ▶ Distributed Transaction Management
- ▶ Logging and Recovery
- ▶ Transaction Model
- ▶ Transaction

Recommended Reading

1. Gawlick D. Processing "hot spots." In Digest of Papers – COMPCON, 1985, pp. 249–251.
2. Gawlick D. and Kinkade D. Varieties of concurrency control in IMS/VS fast path. IEEE Database Eng. Bull., 8(2):3–10, 1985.
3. O'Neil P.E. The escrow transactional method. ACM Trans. Database Syst., 11(4):405–430, 1986.

e-Services

- ▶ Web Services

ETL

- ▶ Extraction, Transformation and Loading

ETL Process

- ▶ Extraction, Transformation and Loading

ETL Tool

- ▶ Extraction, Transformation and Loading

ETL Using Web Data Extraction Techniques

- ▶ Web ETL

European Law in Databases

Michael Carroll
Villanova University, Villanova, PA, USA

Synonyms

Intellectual Property; License

Definition

European law provides three types of legal rights to database owners to control copying: (i) copyright law, (ii) the right to impose contracts or licenses on users, and (iii) a unique statutory right applicable to the non-copyrightable information – such as factual data – in European databases. This entry discusses only the third of these rights. This stand-alone or "sui generis" database right gives a database owner a claim against one who extracts or reuses the whole or a substantial part of the database contents, where "substantial part" is evaluated qualitatively or quantitatively.

Historical Background

Prior to 1996, the legal treatment of databases in Europe varied in the requirements for copyright protection. In 1992, the European Commission proposed that EU copyright law be harmonized with US law by requiring originality for copyright protection to apply. However, the European Commission also proposed that a separate legal right should be granted for un-original databases that require substantial investment to create or update. After further revision to ensure certain user rights, the proposal was adopted as Directive 96/9/EC of the European Parliament and of the Council on the legal protection of databases. The Directive required Member States to harmonize their copyright treatment of databases with US law and to further enact national legislation providing database owners with a new stand-alone legal right to control certain copying of non-copyrightable information in databases. All 25 countries subject to the Directive have complied with this requirement.

An analysis of the effect of the Directive conducted by the European Commission in 2005 concluded that "[w]ith respect to 'non-original' databases, the assumption that more and more layers of IP protection means more innovation and growth appears not to hold up." The report offered policymakers four options: (i) Repeal the whole Database Directive; (ii) Withdraw the sui generis right while leaving protection for creative databases unchanged; (iii) Amend the sui generis provisions in order to clarify their scope; (iv) Maintain the status quo. Comments were solicited from specific stakeholders. There was support for each option and as of January 2008 no legislation was pending.

Foundations

Article 7 of the Directive requires that: "Member States shall provide for a right for the maker of a database which shows that there has been qualitatively and/or quantitatively a substantial investment in either the obtaining, verification or presentation of the contents to prevent extraction and/or reutilization of the whole or of a substantial part, evaluated qualitatively and/or quantitatively, of the contents of that database."

This provision can be broken down into (i) the eligibility criteria and (ii) the scope of the rights granted to those deemed eligible.

Eligibility

Under the Directive, a database is "a collection of independent works, data or other materials arranged in a systematic or methodical way and individually accessible by electronic or other means." To qualify for this database right, the creator, which, unlike in European copyright law, can be a corporation, must have made qualitatively or quantitatively a substantial investment in either the obtaining, verification or presentation of the contents."

Notice that the eligibility criteria allow for multiple rightsholders in the same data. For example, Company *A* may make a substantial investment in collecting data. Company *A* may then sell the data to Company *B*, which makes a substantial investment in verifying the data. Company *B* may then sell the data to Company *C*, which makes a substantial investment in presenting the data.

This database right is initially held by the person or corporation which made the substantial investment, so long as (i) the person is a national or domiciliary of a Member State or (ii) the corporation is formed according to the laws of a Member State and has its registered office or principal place of business within the European Union. The database right lasts for 15 years from the date of publication or, in the case of unpublished databases, from the year of creation.

Scope of Rights – Extraction or Reutilization

Under the Directive, a rightsholder may bring a claim for unauthorized "extraction" or "reutilization" of a "substantial part" of a database.

Extraction is defined broadly to mean "the permanent or temporary transfer of all or a substantial part of the contents of a database to another medium by any means or in any form."

Reutilization is also defined broadly to mean "any form of making available to the public all or a substantial part of the contents of a database by the distribution of copies, by renting, by on-line or other forms of transmission."

A substantial part of the database is to be evaluated quantitatively or qualitatively, which means that even extraction or reuse of a small amount of data may infringe the right if that data has economic value to the rightholder.

Somewhat in conflict with the express language defining the right as covering only substantial parts of the database, Article 7(5) also provides that "repeated and systematic extraction and/or reutilization of insubstantial parts of the contents of the database implying acts which conflict with a normal exploitation of that database or which unreasonably prejudice the legitimate interests of the maker of the database shall not be permitted."

Exceptions and Limitations

The broad rights granted under the Directive are subject to specific exceptions and limitations.

Where copies of the database are sold in material form, such as on a CD ROM, the rightholder's reutilization right is exhausted after the first sale. This means one may resell or otherwise redistribute one's copy of the database without liability. Public lending of such a copy also is not an infringement of the reutilization right.

In addition, the Directive provides three specific exceptions under which a substantial part of a database may be extracted or reutilized without permission:

1. In the case of extraction for private purposes of the contents of a non-electronic database;
2. In the case of extraction for the purposes of illustration for teaching or scientific research, as long as the source is indicated and to the extent justified by the non-commercial purpose to be achieved; and
3. In the case of extraction and/or reutilization for the purposes of public security or an administrative or judicial procedure.

Importantly, under the Directive, the rightsholder may not supplement the protection of the database right with a contractual license that restricts the user's right to use insubstantial parts of the database without authorization.

Key Applications

At the national level, court judgments reflect a range of databases considered eligible for the sui generis right. These include listings of telephone subscribers; databases compiling case-law and legislation; websites containing classified advertisements; catalogs of various sorts; lists of headings of newspaper articles. The European Court of Justice also has accepted a broad reading of what collections of information qualify as a database under the Directive.

The eligibility requirement of "substantial investment" in the database has received varied, and arguably conflicting, interpretations in the courts. Examples of databases that courts have satisfied the criterion include a compilation of several thousand real estate listings (Belgium) and the effort to collect and verify the data for the weekly "Top 10" hit chart of music titles (Germany). Examples of databases that have failed to show substantial investment include a website containing information on building construction (Germany) and a listing of newspaper headlines that were deemed to be a "spin-off" of the primary database of news articles (Netherlands). A related area of divergent judgments concerns copying from on-line databases and Internet-related activities such as "hyper linking" or "deep-linking" using search engines.

Related differences of opinion have emerged with respect to what constitutes an extraction or reutilization of a "substantial part" of the database, particularly with regard to indexing of on-line resources such as news sites by search engines.

Disputes over sports-related data were referred to the European Court of Justice in 2004 to clarify the scope of the Directive. The ECJ rejected claims by sports bodies that organize contests to ownership over the scores or results and other data related to these contests.

Cross-references

► Copyright Issues in Databases
► Licensing and Contracting Issues in Databases

Recommended Reading

1. Comments on future of the Directive. http://circa.europa.eu/Public/irc/markt/markt_consultations/library?l=/copyright_neighbouring/database_consultation&vm=detailed&sb=Title
2. Commission of the European Communities, First evaluation of Directive 96/9/EC on the legal protection of databases, 12 Dec. 2005, at http://ec.europa.eu/internal_market/copyright/docs/databases/evaluation_report_en.pdf
3. Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, OJ L 77, 27.3.1996, pp. 20–28. http://eur-lex.europa.eu/smartapi/cgi/sga_doc?smartapi!celexapi!prod!CELEXnumdoc&lg=en&numdoc=31996L0009&model= guichett
4. Gervais, D.J. The Protection of Databases (2007): Chicago-Kent Law Review, 82:1109–1168, 2007 available at http://works.bepress.com/cgi/viewcontent.cgi?article=1011&context=daniel_gervais.
5. Judgments of the European Court of Justice in respect of the Database Directive, http://ec.europa.eu/internal_market/copyright/prot-databases/jurisprudence_en.htm
6. Maurer, S.M. Across Two Worlds: Database Protection in the United States and Europe, In Intellectual Property and Innovation in the Knowledge–Based Economy, 2001. http://strategis.ic.gc.ca/epic/site/ippd-dppi.nsf/vwapj/13-EN2%20Maurer.pdf/$file/13-EN2%20Maurer.pdf
7. Maurer, S.M., Hugenholtz, B.P., and Onsrud, Harlan J. Europe's Database Experiment, Science 2001, 294(5543): 789–790, 2001.
8. Overview of official documents. http://ec.europa.eu/internal_market/copyright/prot-databases/prot-databases_en.htm
9. Study on the Implementation and Application of Directive 96/9/EC on the Legal Protection of Databases conducted by NautaDulith. http://ec.europa.eu/internal_market/copyright/docs/databases/etd2001b53001e72_en.pdf

Evaluation Forum

► INitiative for the Evaluation of XML retrieval (INEX)

Evaluation in Information Retrieval

► Standard Effectiveness Measures

Evaluation Measures

► Search Engine Metrics

Evaluation Metrics for Structured Text Retrieval

Jovan Pehcevski[1], Benjamin Piwowarski[2]
[1]INRIA Paris-Rocquencourt, Le Chesnay Cedex, France
[2]University of Glasgow, Glasgow, UK

Synonyms

Performance metrics; Evaluation of XML retrieval effectiveness

Definition

An *evaluation metric* is used to evaluate the effectiveness of information retrieval systems and to justify theoretical and/or pragmatic developments of these systems. It consists of a set of measures that follow a common underlying evaluation methodology.

There are many metrics that can be used to evaluate the effectiveness of structured text retrieval systems. These metrics are based on different evaluation assumptions, incorporate different hypotheses of the expected user behavior, and implement their own evaluation methodologies to handle the level of overlap among the units of retrieval.

Historical Background

Over the past 5 years, the initiative for the evaluation of XML retrieval (INEX) has investigated various aspects of structured text retrieval, by particularly focusing on XML retrieval. Major advances, both in terms of approaches to XML retrieval and evaluation of XML retrieval, have been made in the context of INEX. The focus of this entry is on evaluation metrics for XML retrieval, as evaluation metrics for structured text retrieval have thus far been proposed in the context of XML retrieval.

Compared to traditional information retrieval, where whole documents are the retrievable units, information retrieval from XML documents creates additional evaluation challenges. By exploiting the

logical document structure, XML allows for more focused retrieval by identifying information units (or XML elements) as answers to user queries. Due to the underlying XML hierarchy, in addition to finding the most specific elements that at the same time exhaustively cover the user's information need, an XML retrieval system needs to also determine the appropriate level of answer granularity to return to the user. The *overlap problem* of having multiple nested elements, each containing identical textual information, can have a huge impact on XML retrieval evaluation [6].

Traditional information retrieval evaluation measures (such as recall and precision) mostly assume that the relevance of an information unit (e.g., a document) is binary and independent of the relevance of other information units, and that the user has access to only one information unit at a time. Furthermore, they also assume that the information units are approximately equally sized.

These two assumptions do not hold in XML retrieval, where the information units are nested elements of very different sizes. As nested elements share parts of the same information, an evaluation metric for XML retrieval can no longer assume than the relevance of elements is independent. Moreover, since users can access different parts of an XML document, it also can no longer be assumed that they will have access to only one element at a time. Each of the evaluation metrics for XML retrieval supports the above assumptions to a different extent.

Another limitation of the traditional information retrieval metrics is that they are not adapted to the evaluation of specific retrieval tasks, which could use more advanced ways of presenting results that arise naturally when dealing with XML documents. For example, one task in XML retrieval is to present the retrieved elements by their containing documents, allowing for an easier identification of the relevant information within each document.

INEX has been used as an arena to investigate the behavior of a variety of evaluation metrics for XML retrieval. Most of them are extensions of traditional information retrieval metrics, namely precision-recall and cumulated gain. Precision-recall is a bi-dimensional metric that captures the concentration and the number of relevant documents retrieved by an information retrieval system. An alternative definition of this metric calculates precision at a given recall level ℓ (between 0 and 100%) as the probability that a retrieved document is relevant, provided that a user wants to see ℓ percent of the relevant documents that exist for the topic of interest [12]. The precision-recall metric was the first one extended for the purposes of XML retrieval evaluation.

The cumulated gain (CG) metric [2] relies on the idea that each retrieved document corresponds to a gain for the user, where the gain is a value between 0 and 1. The metric then simply computes the CG at a given rank k as a sum of the gains for the documents retrieved between the first rank and rank k. When normalized, the CG value is somewhat similar to recall, and it is also possible to construct an equivalent of precision for CG. The importance of extending this metric for XML retrieval lies in the fact that it allows for non-binary relevance, which means it can capture elements of varying sizes and granularity.

Foundations

A common notation is used throughout this document to describe the formulae of the different evaluation metrics for XML retrieval. The notation is presented in Table 1. It is also assumed that any XML element can be represented as a textual segment that spans the text corresponding to that XML element. This conceptual representation is practical, as it is possible to define the intersection, the union, the inclusion, and the size of any two segments.

Evaluation Metrics for Structured Text Retrieval.

Table 1. Common notations used to describe formulae of XML retrieval metrics

Notation	Short description
e	An XML element
e_i	The ith XML element in the list
$spe(e)$	The (normalized) specificity
$exh(e)$	The (normalized) exhaustivity
$q(e)$	A quantization function
$\mathfrak{I}$	The set of ideal elements
$\mathcal{L}$	The ideal ranked list of elements
ℓ	Arbitrary recall level
size(e)	The size of element e, usually in number of characters
overlap(i)	The level of overlap between the ith element of the list and the previously returned elements

Evaluation Concepts

In XML retrieval, the commonly used ad hoc retrieval task simulates how a digital library is typically used, where information residing in a static set of XML documents is retrieved using a new set of topics. Different sub-tasks can be distinguished within the broad ad hoc retrieval task.

XML Retrieval Tasks

The main XML retrieval tasks, considered to be subtasks of the main INEX ad hoc retrieval task, are:

- Thorough, where XML retrieval systems are required to estimate the relevance of a retrieved element and return a ranked list of all the overlapping relevant elements.
- Focused, where the returned ranked list consists of non-overlapping relevant elements.
- Relevant in context (RiC), where systems are required to return a ranked list of relevant articles, where for each article a set of non-overlapping relevant elements needs to be correctly identified.
- Best in context (BiC), where the systems are required to return a ranked list of relevant articles, where for each article the best entry point for starting to read the relevant information within the article needs to be correctly identified.

User Behavior

The evaluation metrics typically model a sequential user browsing behavior: given a ranked list of answer elements, users start from the beginning of the list and inspect one element at a time, until either all the elements in the list have been inspected, or users had stopped inspecting the list since their information needs were fully satisfied. However, while inspecting a ranked list of elements, users of an XML retrieval system could also have access to other structurally related elements, or indeed could be able to inspect the *context* where the answer elements reside (which may be supported by features such as browsing, scrolling, or table of contents).

Accordingly, in addition to modeling the sequential user model, the evaluation metrics should also be able to model various user browsing behaviors.

Relevance Dimensions

The *relevance* of a retrieved XML element to a query can be described in many ways. It is therefore necessary to define a relevance scale that can be used by the evaluation metrics. Traditional information retrieval usually uses a binary relevance scale, while in XML retrieval there is a multi-graded (or continuous) relevance scale that uses the following two relevance dimensions:

- Exhaustivity (denoted exh), which shows the extent to which an XML element covers aspects of the information need.
- Specificity (denoted spe), which shows the extent to which an XML element is focused on the information need.

The two relevance dimensions have evolved over the years (readers are referred to the *relevance* definitional entry for more details). For simplicity, it will be assumed that each relevance dimension uses a continuous relevance scale with values between 0 and 1. For example, the four-graded relevance scale used by the two dimensions in INEX from 2002 until 2004 can be mapped onto the values 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1.

The normalized exhaustivity and *specificity* of an XML element e are respectively denoted as $exh(e)$ and $spe(e)$. They can take values between 0 and 1.

Quantization

Quantization is the process of transforming the values obtained from the two relevance dimensions into a single normalized relevance score (which again takes values between 0 and 1). It is used to represent the extent to which the retrieved element is relevant.

For example, the strict quantization function can be used to measure the XML retrieval performance when only highly relevant elements are targets of retrieval, while the generalized quantization function can be used to measure the performance when elements with multiple degrees of relevance are targets of retrieval:

$$q_{\text{strict}}(e) = \begin{cases} 1 & \text{if } exh(e) = spe(e) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$q_{\text{gen}}(e) = exh(e) \times spe(e)$$

The strict quantization can therefore be used to reward systems that only retrieve elements that are fully exhaustive and specific, while the generalized quantization rewards systems that retrieve elements with multiple relevance degrees.

Ideality

The concept of ideality emerged in XML retrieval as a concept that is used to distinguish those among all judged relevant elements that users would prefer to see as answers. For example, in order to distinguish between the intrinsic relevance of a paragraph from the inherited relevance of its containing section, it could be said that, even though both elements are relevant, only the paragraph is *ideal*. By definition, an ideal element is always relevant but the reverse is true only in traditional information retrieval.

Ideal elements, unlike relevant elements, can be assumed to be independent. Note that this assumption is similar to the independence of document relevance in traditional information retrieval; that is, ideal elements, as documents, can overlap *conceptually* (they can contain same answers to the underlying information need) as long as they do not overlap *physically*. In XML retrieval, this assumption implies that ideal elements cannot be nested. Note that ideality can be extended to more general units than elements, namely the passages.

Construction of Ideal Sets and Lists

To construct a set $\mathfrak{I}$ of ideal elements, one has to make hypotheses about the underlying retrieval task and the expected user behavior [5].

One example of methodology for identifying the ideal elements is as follows [3]: given any two elements on a relevant path, the element with the higher quantized score is first selected. A *relevant path* is a path in the document tree that starts from the document element and ends with a relevant element that either does not contain other elements, or contains only irrelevant elements. If the two element scores are equal, the one deeper in the tree is chosen. The procedure is applied recursively to all overlapping pairs of elements along a relevant path until only one element remains. It is important that the methodology for identifying ideal elements closely reflects the expected user behavior, since it has been shown that the choice of methodology can have a dramatic impact on XML retrieval evaluation [3].

Given a set of ideal elements, and an evaluation metric that uses that set, it is then possible to construct an ideal list $\mathcal{L}$ of retrieved elements that maximises the metric score at each rank cutoff.

Near Misses and Overlap

Support of near misses is an important aspect that needs to be considered by the evaluation metrics for XML retrieval. *Near misses* are elements close to an ideal element, which act as entry points leading to one or more ideal elements. It is generally admitted that systems that retrieve near misses should be rewarded by the evaluation metrics, but to a lesser extent than when ideal elements are retrieved [4].

Early attempts that extended the traditional information retrieval metrics to support XML retrieval rewarded near misses by assigning partial scores to the elements nearby an ideal one [1,6]. However, this implies that systems that return only ideal elements will never achieve a 100% recall, since both ideal elements and near misses have to be returned to achieve this level of recall [10].

Moreover, these metric extensions are commonly considered to be "overlap positive" [14], which means that they reward systems for retrieving twice the same ideal element, either directly or indirectly, and that the total reward for retrieving that ideal element increases with the number of times it is retrieved. To cater for this problem, overlap neutral and/or negative evaluation metrics have since been developed [1,5].

It is therefore important to be able to compute the degree of overlap between an element e_i and other elements previously retrieved in the ranked list $(e_1,...,e_i)$. A commonly adopted measure is the percentage of text in common between the element and the other previously retrieved elements:

$$\text{overlap}(i) = \frac{\text{size}\left(e_i \cap \bigcup_{j=1}^{i-1} e_j\right)}{\text{size}(e_i)} \qquad (1)$$

The overlap function equals 0 when there is no overlap between the element e_i and any of the previously retrieved elements e_j, and equals 1 when there is full overlap (i.e., either all its descendants or one of its ancestors have been retrieved). A value between 0 and 1 denotes intermediate possibilities.

Metric Properties

An evaluation metric for XML retrieval should provide a support for the following properties.

- *Faithfulness*. The metric should measure what it is supposed to measure (*fidelity*) and it should be *reliable* enough so that its evaluation results can be trusted.
- *Interpretation*. The outcome of the evaluation metric should be easy to interpret.

- *Recall/precision*. The metric should capture both *recall* and *precision*, as they are complementary dimensions whose importance have been recognized in traditional information retrieval (some retrieval tasks put more focus on recall while others prefer precision).
- *Ideality*. The metric should support the notion of ideal elements.
- *Near misses*. The metric should be able to properly handle near misses.
- *Overlap*. The metric should properly handle the overlap among retrieved and judged elements.
- *Ideality graded scale*. The metric should be able to support multi-graded or continuous scales, in order to distinguish the ideality of two elements.
- *User models and retrieval tasks*. The metric should be able to model different *user behaviors* and support different *retrieval tasks*, since XML retrieval systems support a variety of features that allow information access.

Table 2 summarizes the above metric properties and provides an overview of the extent to which each of the evaluation metrics for XML retrieval (described in the next section) provides a support for them.

Evaluation Metrics

This section presents the different evaluation metrics that were proposed so far in XML retrieval.

The inex_eval Metric

For 3 years from 2002, the `inex_eval` metric [1] was used as the official INEX metric to evaluate the effectiveness of XML retrieval systems. This metric supports *weak ordering* of elements in the answer list [12], where one or more elements are assigned identical retrieval status values by an XML retrieval system. For simplicity, the discussion is restricted to the case where elements are fully ordered.

The `inex_eval` metric assumes that the degree (or probability) of relevance of an element e is directly given by the quantization function $q(e)$. Its degree

Evaluation Metrics for Structured Text Retrieval. Table 2. Metric properties, and the extent to which each of the XML retrieval metrics supports them. In the table, "y" stands for yes, "n" for no, "i" for indirect, "+" for overlap positive, "−" for overlap negative, and "=" for overlap neutral. The question mark "?" signifies unclear or not demonstrated property

Property \ Metric	inex_eval	inex_eval_ng	nXCG	ep/gr	T2I	GR	PRUM	EPRUM	HiXEval
Research publication	[1]	[1]	[5]	[5]	[13]	[10]	[11]	[9]	[7]
INEX metric (years)	02–04	03	05–06	05–06				06	07
Faithfulness	?	?	y	y	?	y	y	y	y
Interpretation	n[a]	n[a]	y[b]	y[b]	y	y	y	y	y[b]
Recall	y	y	y	y	y	y	y	y	y
Precision	y	y	n	y	y	y	y	y	y
Near misses	i	i	y	y	y	y	y	y	y
Overlap	+	-	=	=	=	=	=	=	=
Ideality	n	n	y	y	y	y	y	y	y[c]
Ideality graded scale	n/a	n/a	y	y	n	y	n	y	y[d]
Explicit user model	n	n	n	n	y	y	y	y	n
XML Retrieval Tasks									
Thorough	y	y	y	y	?	y	y	y	y
Focused	?	?	y	y	y	y	y	y	y
RiC	i	i	i	i	i	y	y	y	i
BiC	i	i	i	i	i	y	y	y	i

[a]But for some special cases.
[b]With parameter α set to 0 or 1.
[c]Highlighted passages are the ideal units in the case of HiXEval.
[d]The ideality of an element is fixed and directly proportional to the amount of highlighted text.

of non-relevance can be symmetrically defined as $(1 - q(e_i))$. At a given rank k, it is then possible to define the expected number of relevant (resp. non-relevant) $R(k)$ (resp. $I(k)$) elements as follows:

$$R(k) = \sum_{i \le k} q(e_i) \quad I(k) = \sum_{i \le k} (1 - q(e_i))$$

For a given recall level ℓ, the *Precall* [12] metric estimates the probability that a retrieved element is relevant to a topic (assuming that a user wants to find ℓ% of the relevant elements in the collection, or equivalently $\ell \cdot N$ relevant elements). If k_ℓ is the smallest rank k for which $R(k)$ is greater or equal to $\ell \cdot N$, then precision is defined as follows:

$$\text{Precision}(\ell) = \frac{\text{number of seen relevant units}}{\text{expected search length}} = \frac{\ell \cdot N}{k_\ell} \tag{2}$$

where N is assumed to be the expectation of the total number of relevant elements that can be found for an INEX topic, i.e., $N = \sum_e q(e)$ across all the elements of the collection.

Beyond the lack of support for various XML retrieval tasks, the main weakness of the `inex_eval` metric is that one has to choose (with the quantization function) whether the metric should allow near misses or should be overlap neutral – both are not possible. To support overlap, it is possible to compute a set of ideal elements by setting the normalized quantization scores of non-ideal elements to 0, thus not rewarding near misses. To reward near misses, the quantization function should give a non-zero values for elements nearby the ideal elements, but then the system will get fully rewarded only if it returns both the ideal and the other relevant elements. Another problematic issue is the use of non-binary relevance values inside the `inex_eval` formula shown in (2), which makes the metric ill-defined from a theoretical point of view.

The inex_eval_ng Metric

The `inex_eval_ng` metric was proposed as an alternative evaluation metric at INEX 2003 [1]. Here, the two relevance dimensions, *exhaustivity* and *specificity*, are interpreted within an *ideal concept space*, and each of the two dimensions is considered separately while calculating recall and precision scores. There are two variants of this metric, which differ depending on whether overlap among retrieved elements is penalized or not: `inex_eval_ng(o)`, which penalises overlap among retrieved elements; and `inex_eval_ng(s)`, which allows overlap among retrieved elements. Unlike the `inex_eval` metric, this metric directly incorporates element sizes in their relevance definitions.

With `inex_eval_ng(o)`, precision and recall at rank k are calculated as follows:

$$\text{Precision}(k) = \frac{\sum_{i=1}^{k} spe(e_i) \cdot \text{size}(e_i) \cdot (1 - \text{overlap}(i))}{\sum_{i=1}^{k} \text{size}(e_i) \cdot (1 - \text{overlap}(i))}$$

$$\text{Recall}(k) = \frac{\sum_{i=1}^{k} exh(e_i) \cdot (1 - \text{overlap}(i))}{\sum_{i=1}^{N} exh(e_i)}$$

With `inex_eval_ng(s)`, recall and precision are calculated in the same way as above, except that here the overlap function is replaced by the constant 0 (by which overlap among the retrieved elements is not penalized).

The `inex_eval_ng` metric has an advantage over `inex_eval`, namely the fact that it is possible to penalise overlap. However, due to the fact that it ignores the ideality concept, the metric has been shown to be very unstable if one changes the order of elements in the list, in particular the order of two nested elements [11]. Moreover, `inex_eval_ng` treats the two relevance dimensions in isolation by producing separate evaluation scores, which is of particular concern in evaluation scenarios where combinations of values from the two relevance dimensions are needed to reliably determine the preferable retrieval elements.

The XCG Metrics

In 2005 and 2006, the eXtended cumulated gain (XCG) metrics [5] were adopted as official INEX metrics. The XCG metrics are extensions of the cumulated gain metrics initially used in document retrieval [2].

Gain and Overlap

When the cumulated gain (CG) based metrics are applied to XML retrieval, they follow the assumption that the user will read the whole retrieved element, and not any of its preceding or following elements. An element is partially seen if one or more of its descendants have

already been retrieved ($0 < \text{overlap}(i) < 1$), while it is completely seen if any of its ancestors have been retrieved ($\text{overlap}(i) = 1$).

To consider the level of overlap among judged relevant elements, the XCG metrics make use of an ideal set of elements, also known as the *ideal recall base*. To consider the level of overlap among the retrieved elements in the answer list, the XCG metrics implement the following result-list dependent relevance value (or gain) function:

$$\text{gain}(i) = \begin{cases} q(e_i) & \text{if overlap}(i) = 0 \\ (1-\alpha)\cdot q(e_i) & \text{if overlap}(i) = 1 \\ \alpha \cdot \dfrac{\sum_{j/e_j \subseteq e_i} \text{gain}(j)\cdot \text{size}(e_j)}{\text{size}(e_i)} + (1-\alpha)\cdot q(e_i) & \text{otherwise} \end{cases} \quad (3)$$

The parameter α influences the extent to which the level of overlap among the retrieved elements is considered. For example, with α set to 1 (focused task), the gain function returns 0 for a previously fully seen element, reflecting the fact that an overlapping (and thus redundant) element does not bring any retrieval value in evaluation. Conversely, the level of overlap among the retrieved elements is ignored with α set to 0 (thorough task).

The gain formula cannot guarantee that the sum of the gain values obtained for descendants of an ideal element are smaller than the ideal element gain, and so it is necessary to "normalize" the gain value by forcing an upper gain bound [5].

XCG Metrics

Given a ranked list of elements for an INEX topic, the cumulated gain at rank k, denoted as $XCG(k)$, is computed as the sum of the normalized element gain values up to and including that rank:

$$XCG(k) = \sum_{i=1}^{k} \text{gain}(i) \quad (4)$$

Two XCG metrics used as official XML retrieval metrics at INEX in 2005 and 2006 are `nXCG` and `ep/gr`. The `nXCG` metric is a normalized version of $XCG(k)$, defined as the ratio between the gain values obtained for the evaluated list to the gain values obtained for the ideal list.

The `ep/gr` metric was defined as an extension of `nXCG` in order to average performances over runs and to define an equivalent of precision. It consists of two measures: effort-precision *ep*, and gain-recall *gr*.

The gain-recall *gr*, calculated at the rank k, is defined as:

$$gr[k] = \frac{XCG[k]}{\sum_{x \in \mathfrak{I}} gain(x)} \quad (5)$$

The effort-precision *ep* is defined as the amount of relative effort (measured as the number of visited ranks) a user is required to spend compared to the effort they could have spent while inspecting an optimal ranking. It is calculated at a cumulated gain level achieved at rank k and is defined as:

$$ep[k] = \frac{\min\{i | XCG_{\mathfrak{L}}(i) \geq XCG(k)\}}{k} \quad (6)$$

where the indice $\mathfrak{L}$ means that the score is evaluated with respect to an ideal list of relevant elements. An *ep* score of 1 reflects an ideal performance, in which case the user made the minimum necessary effort (computed in number of ranks) to reach that particular cumulated gain. An ep/gr curve can then be computed by taking pairs $(gr[k], ep[k])$ for varying rank k values.

The XCG metrics have advantages over `inex_eval` and `inex_eval_ng`, since its use of an ideal list ensures that the metric is overlap neutral. It also properly handles near misses by the use of an appropriate quantization function. However, the construction of the ideal set of elements relies on heuristics [3]. Other problems of the metric is that the gain is difficult to interpret for values of α other than 0 or 1, which also makes the outcome of the metric somewhat difficult to interpret.

The T2I metric

The tolerance to irrelevance (T2I) metric [13] relies on the same evaluation assumptions as `inex_eval`, but includes a different user model more suited to XML documents. The underlying user model is based on the intuition that a user processes the retrieved list of elements until their tolerance to irrelevance have been reached (or until they found a relevant element), at which point the user proceeds to the next system result. The T2I metric has only been theoretically proposed, and is yet to be implemented and evaluated.

The (E)PRUM and GR Metrics

The expected precision recall with user modeling (EPRUM) metric [9], which was used as an alternative evaluation metric at INEX in 2005 and as one of the official ones in 2006, extends the traditional definitions of precision and recall to model a variety of user behaviors. EPRUM is unique among all the INEX metrics in that it stochastically defines the user browsing behavior. It is the last defined within a set of three metrics, the previous one being GR (generalized recall) and PRUM (precision recall with user modeling).

The User Model

From a retrieved element, the user can navigate using the corpus structure. The *context* of a list item is defined as the set of elements that can be reached through navigation from it. This includes the pointed elements but also the context of the pointed elements (siblings, ancestors, etc.). To model the user behavior inside the context, the three metrics rely on a set of probabilities on simple events of the form "navigating from a list item to an element in the corpus". The probabilities of navigating from rank j in the list to an element x can be set to values estimated by any adequate method and is denoted $\mathrm{P}(j \rightsquigarrow x)$. When a user is over with this exploration, they *consult* the next entry of the list and repeat the process until their information needs are satisfied. Note that this user model is general enough so as to cope with all INEX tasks, since there is no constraint on how a rank is defined.

Users *see* an element when they navigate to it from another element or from the list, and they *discover* an element if they see it for the first time. The distinction between "seen" and "discovered" is important because the system is rewarded only when elements are discovered. The probability that the user discovers f ideal elements when consulting ranks between 1 and k included is then given by:

$$\mathrm{P}(F_k = f) = \sum_{\substack{A \subseteq \mathfrak{I} \\ |A| = f}} \prod_{x \in A} \mathrm{P}(x \in S_k) \prod_{x \in \mathfrak{I} \setminus A} \mathrm{P}(x \notin S_k) \quad (7)$$

where S_k is the set of all elements seen by the user who has consulted ranks 1 to k. The probability that an element was seen is computed with $P(x \in S_k) = 1 - \prod_{j=1}^{k}(1 - P(j \rightsquigarrow x))$.

GR, PRUM and EPRUM

The GR metric is a generalization of recall with the above specified user model. It simply estimates the expected number of discovered elements at a given rank k, and divides it by the expected number of ideal elements in the database in order to get a normalized value. PRUM is defined as the probability that a consulted list item leads the user to discover an ideal element. Its most important limitation is that it does not handle well non-binary assessments.

EPRUM defines precision based on the comparison of two minimum values: the minimum rank that achieves the specified recall over *all* the possible lists and over the *evaluated* list. For a given recall level ℓ, precision is thus defined as the percentage of effort (in minimum number of consulted ranks) a user would have to make when consulting an ideal list with respect to the effort when consulting the evaluated list:

$$\mathrm{Precision}(\ell) = \mathbb{E}\left[\frac{\begin{array}{c}\text{Minimum number of consulted}\\ \text{list items for achieving a recall } \ell \text{ over all lists}\end{array}}{\begin{array}{c}\text{Minimum number of consulted list items}\\ \text{for achieving a recall } \ell \text{ over the evaluated list}\end{array}}\right]$$

where the assumption is that when the user cannot achieve recall ℓ in the evaluated list, then the minimum number of consulted list items for the evaluated list is infinite (this assumption is the same as in traditional information retrieval). This measure is an extension of the standard precision-recall metric.

It is similarly possible to extend the traditional definition of precision at a given rank k. If the expected recall of the evaluated list at rank k is r_k, then precision at rank k is defined as the ratio of the minimum number of consulted list items over all possible lists to achieve recall r_k to the number of consulted ranks k.

The EPRUM metric solves some problems of PRUM and substantially reduces its complexity. It also allows proper handling of graded ideality. The advantages of EPRUM metric are the fact that it handles all the INEX tasks through its user model parameters, that the user model is very flexible (for example allowing to reward near misses that are not direct ancestors or descendant of an ideal element), and that the outcome of the metric can easily be interpreted. However, like the XCG metrics, it assumes that the ideal set of elements and the ideal list of retrieved elements

can easily be determined, which is shown to be not as straightforward in XML retrieval [3].

The HiXEval Metric

Since 2005, a highlighting assessment procedure is used at INEX to gather relevance assessments for the XML retrieval topics. In this procedure, assessors from the participating groups are asked to highlight sentences representing the relevant information in a pooled set of retrieved documents. To measure the extent to which an XML retrieval system returns relevant information, INEX started to employ evaluation metrics based on the `HiXEval` metric [7,8]. This is motivated by the need to directly exploit the INEX highlighting assessment procedure, and it also leads to evaluation metrics that are natural extensions of the well-established metrics used in traditional information retrieval.

`HiXEval` only considers the specificity relevance dimension, and it credits systems for retrieving elements that contain as much highlighted (relevant) text as possible, without also containing a substantial amount of non-relevant text. So, instead of counting the number of relevant elements retrieved, `HiXEval` measures the amount of relevant text retrieved. Like the XCG metrics, it makes the assumption that the user will read the whole retrieved element, and not any of its preceding or following elements. An element is partially seen by the user if one or more of its descendants have already been retrieved, while it is completely seen if any of its ancestors have been retrieved.

Let $\text{rsize}(e_i)$ be the amount of highlighted (relevant) text contained by an element e retrieved at rank i, so that if there is no highlighted text in the element, $\text{rsize}(e_i) = 0$. (Note that $\text{rsize}(e_i)$ can also be represented as: $\text{rsize}(e_i) = spe(e_i) \cdot \text{size}(e_i)$.) To measure the value of retrieving relevant text from e_i, the relevance value function $\text{rval}(i)$ is defined as follows:

$$\text{rval}(i) = \begin{cases} \text{rsize}(e_i) & \text{if overlap}(i) = 0 \\ (1-\alpha) \cdot \text{rsize}(e_i) & \text{if overlap}(i) = 1 \\ \text{rsize}(e_i) - \alpha. \sum\limits_{j/e_j \subseteq e_i} \text{rval}(j) & \text{otherwise} \end{cases} \tag{8}$$

As with the XCG metrics, the parameter α is a weighting factor that represents the importance of retrieving non-overlapping elements in the ranked list.

Precision and recall at a rank k are defined as follows:

$$\text{Precision}(k) = \frac{\sum_{i=1}^{k} \text{rval}(i)}{\sum_{i=1}^{k} \text{size}(e_i)}$$

$$\text{Recall}(k) = \frac{1}{Trel} \cdot \sum_{i=1}^{k} \text{rval}(i)$$

In the above equation, *Trel* represents the total amount of highlighted relevant text for an INEX topic. Depending on the XML retrieval task, different *Trel* values are used by the metric. For example, for the focused task *Trel* is the total number of highlighted characters across all *documents*. This means that the total amount of highlighted relevant text for the topic represents the sum of the sizes of the (non-overlapping) highlighted passages contained by all the relevant documents. Conversely, for the thorough task *Trel* is the total number of highlighted characters across all *elements*. For this task, the total amount of highlighted relevant text for the topic represents the sum of the sizes of the (overlapping) highlighted passages contained by all the relevant elements.

The precision and recall scores can be combined in a single score using the standard F-measure (their harmonic mean). By comparing the F-measure scores obtained from different XML retrieval systems, it would be possible to see which system is more capable of retrieving as much relevant information as possible, without also retrieving a substantial amount of non-relevant information.

HiXEval has the advantage of using a naturally defined ideal unit, namely a highlighted passage, and thus overcomes the problem of defining a set of ideal elements in an arbitrary way. One shortcoming of this metric is that it makes the assumption that the degree of ideality of a passage is directly proportional to the passage size. It also shares the same issue identified with the XCG metrics that, with α values different from 0 or 1, the interpretation of the output of the $\text{rval}(i)$ function is not very straightforward.

Key Applications

Web Search

Due to the increasing adoption of XML on the World Wide Web, information retrieval from XML document

collections has the potential to be used in many Web application scenarios. Accurate and reliable evaluation of XML retrieval effectiveness is very important for improving the usability of Web search, especially if the evaluation captures the extent to which XML retrieval can be adapted to a particular retrieval task or a user model. This could certainly justify the increasing usage of XML in the ever-growing number of interactive Web search systems.

Digital Libraries

Reliable evaluation of XML retrieval effectiveness is also important for improving information retrieval from digital libraries, especially since there is a large amount of structured (XML) information that is increasingly stored in modern digital libraries.

URL to Code

EvalJ project: http://evalj.sourceforge.net

Cross-references

► INitiative for the Evaluation of XML Retrieval
► XML Retrieval

Recommended Reading

1. Gövert N., Fuhr N., Lalmas M., and Kazai G. Evaluating the effectiveness of content-oriented XML retrieval methods. Inform. Ret., 9(6):699–722, 2006.
2. Järvelin K. and Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans. Inform. Syst., 20(4):422–446, 2002.
3. Kazai G. Choosing an ideal recall-base for the evaluation of the Focused task: Sensitivity analysis of the XCG evaluation measures. In Proc. Comparative Evaluation of XML Information Retrieval Systems: Fifth Workshop of the INitiative for the Evaluation of XML Retrieval, 2007, pp. 35–44.
4. Kazai G. and Lalmas M. Notes on what to measure in INEX. In Proc. INEX 2005 Workshop on Element Retrieval Methodology, 2005, pp. 22–38.
5. Kazai G. and Lalmas M. eXtended Cumulated Gain measures for the evaluation of content-oriented XML retrieval. ACM Trans. Inform. Syst., 24(4):503–542, 2006.
6. Kazai G., Lalmas M., and de Vries A.P. The overlap problem in content-oriented XML retrieval evaluation. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 72–79.
7. Pehcevski J. Evaluation of Effective XML Information Retrieval. Ph.D. thesis, RMIT University, Melbourne, Australia, 2006.
8. Pehcevski J. and Thom J.A. HiXEval: Highlighting XML retrieval evaluation. In Advances in XML Information Retrieval and Evaluation: Proc. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval, 2006, pp. 43–57.
9. Piwowarski B. and Dupret G. Evaluation in (XML) information retrieval: Expected Precision-recall with user modelling (EPRUM). In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 260–267.
10. Piwowarski B. and Gallinari P. Expected Ratio of Relevant Units: A Measure for Structured Information Retrieval. In INEX 2003 Workshop Proceedings, 2003, pp. 158–166.
11. Piwowarski B., Gallinari P., and Dupret G. Precision recall with user modelling (PRUM): Application to structured information retrieval. ACM Trans. Inform. Syst., 25(1):1–37, 2007.
12. Raghavan V., Bollmann P., and Jung G. A critical investigation of recall and precision. ACM Trans. Inform. Syst., 7(3): 205–229, 1989.
13. de Vries A., Kazai G., and Lalmas M. Tolerance to Irrelevance: A User-effort Evaluation of Retrieval Systems without Predefined Retrieval Unit. In Proceedings of RIAO 2004, 2004, pp. 463–473.
14. Woodley A. and Geva S. XCG Overlap at INEX 2004. In INEX 2005 Workshop Pre-Proceedings. 2005, pp. 25–39.

Evaluation of Fuzzy Queries Over Multimedia Systems

► Top-k Selection Queries on Multimedia Datasets

Evaluation of Relational Operators

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Query evaluation; Query processing

Definition

A query usually consists of several relational operators. The corresponding physical query plan is composed of several physical operators. Generally speaking, a physical operator is an implementation of a relational operator. For each relational operator, there are several alternative algorithms (physical operators) for its implementation and there is no universally superior one. So choosing physical operators wisely is crucial for query performance. Which physical operator is the best depends on several factors, such as the sizes of the input relations, the existing sorting orders of the input relations, the existing indexes/materialized views, the buffer replacement policy, and the size of the available buffer pool, etc.

Historical Background

The relational operators serve as the building blocks for query processing. Their implementation techniques have been extensively studied ever since the first relational DBMS was built.

Foundations

Efficient evaluation of relational operators is required to provide good query performance. There are usually different evaluation algorithms for a given relational operator. Each evaluation algorithm is implemented by a physical operator. There can be more than one physical query plan for a query. The query optimizer considers various physical query plans and chooses the cheapest plan in a cost-based fashion.

Pipelined Query Execution

Database systems usually employ a demand-pull query execution model. Each query plan consists of a pipeline of physical operators. In this model, each operator supports a group of three functions that allows a parent operator to get the result *one* tuple at a time.

1. *Open*. The open() function initializes the state of the iterator by allocating buffers for its inputs and output, and is also used to pass in arguments such as selection predicates that modify the behavior of the operator.
2. *Next*. The next() function calls the next() function on each input node recursively and processes the input tuples until one output tuple is generated. The state of the operator is updated to keep track of how much input has been consumed.
3. *Close*. When all output tuples have been produced through repeated calls to the next() function, the close() function deallocates the state information and performs final housekeeping.

The iterator interface supports pipelining of results naturally. The decision to pipeline or materialize input tuples is encapsulated in the operator-specific code that processes input tuples. Pipelining requires much less space for intermediate results during execution. The demand-pull pipelined query execution model can be executed by a single process or thread. This approach has several advantages such as avoiding inter-process communication between operators, avoiding process synchronization and scheduling, minimizing data copies, keeping only the current data items in memory, and performing lazy operator evaluation.

Selection

Given a selection of the form $\sigma_{R.a\ \theta\ c}(R)$, the algorithms for selection use either scanning or indexing. The condition $R.a\ \theta\ c$ is also called the selection predicate. See Fig. 1.

Selection Based on Scanning

If there is no secondary index on $R.a$ and the relation R is not sorted on $R.a$, the only choice of evaluation is to scan the entire relation. For each tuple, the predicate is evaluated and the tuple is added to the result if the predicate is satisfied.

Selection Based on Indexing

If there is no secondary index on $R.a$ but relation R is sorted on $R.a$, relation R is said to be clustered on $R.a$ and can be viewed as a clustered index itself. A binary search can be used to locate the first tuple that satisfies the selection predicate. If the predicate is a range predicate, for example, $R.a > 1$, the rest of the tuples can be retrieved by scanning R from this location until the predicate is no longer satisfied.

If there is a secondary index (usually a B^+ tree) on $R.a$, an index lookup can be performed to locate the first index entry pointing to a qualifying tuple. Then the leaf pages of the index are scanned to retrieve all entries with the key value satisfying the predicate. Following each of these entries, the corresponding R tuple is then retrieved and added to the result.

If an index contains all the columns that are required by the query, the final R tuple retrieval can be skipped. In this scenario, the selection is done by an index only operation.

In summary, which strategy is the best depends on the index availability, whether the index is clustered or nonclustered, and the selectivity of the selection predicate.

Projection

In general, the projection is of the form $\pi_{a,\ b,\ c}(R)$. Some projected columns may be computed from attributes in R, for example, $\pi_{R.a\ +\ R.\ b,\ R.\ c}(R)$. See Fig. 2. To implement projection, unwanted attributes (i.e.,

```
SELECT *
FROM R
WHERE R.a > 1
```

Evaluation of Relational Operators. Figure 1. Selection query.

those not specified in the projection) are removed and expressions, if any, are computed on the fly.

Joins

The join operation, $R \bowtie_{R.r\ \theta\ S.s} S$, is one of the most useful relational operations and is the primary means of combining information from more than one relation. The condition $R.r\theta S.s$ is called the join predicate.

A join operation can always be implemented as a cross-product followed by selections and projections. See Fig. 3. However, a cross-product generates much larger results and thus can be quite inefficient. It is very important to recognize joins and implement them without materialized the underlying cross-product.

Join operations have received extensive research over years. There are several alternative techniques for implementing joins. Choosing the right join operation and the correct join order plays a major role in finding an optimal physical plan. The decision depends on various factors, such as the size of input relations, the amount of available memory, and available alternative access methods, etc.

Nested Loops Join

Nested loops join is the simplest join algorithm. The algorithm starts with reading the *outer* relation R, and for each tuple $\mathcal{R} \in R$, the *inner* relation S is checked and matching tuples are added to the result.

```
Algorithm 1: Nested Loops Join: R⋈pred (r,s) S
foreach R ∈ R do
    foreach S ∈ S do
        if pred (R.r, S.s) then
            add {R, S} to result
        end
    end
end
```

```
SELECT R.a + R.b, R.c
FROM R
```

Evaluation of Relational Operators. Figure 2. Projection query.

```
SELECT *
FROM R, S
WHERE R.r = S.s
```

Evaluation of Relational Operators. Figure 3. Join query.

One advantage of nested loops join is that it can handle any kind of join predicate, unlike sort-merge join and hash join which mainly deal with an equality join predicate. In order to effectively utilize buffer pages and available indexes, there are two variations of nested loops join.

Block Nested Loops Join Suppose that the memory can hold B buffer pages. If there is enough memory to hold the smaller relation, say R, with at least two extra buffer pages left, the optimal approach is to read in the smaller relation R and to use one extra page as an input buffer to read in the larger relation S and one extra buffer page as an output buffer.

If there is not enough memory to hold the smaller relation, the best approach is to break the outer relation R into *blocks* of $B - 2$ pages each and scan the whole inner relation S for each block of R. As described before, one extra page is used as an input buffer and the other as an output buffer. In this case, the outer relation R is scanned only once while the inner relation S is scanned multiple times.

Index Nested Loops Join If one of the two relations has an index on the join attribute, an index nested loops join can be used to take advantage of the index. Suppose the relation S has a suitable index on the join attribute $S.s$. For each tuple $\mathcal{R} \in R$, the index can be used to retrieve all matching tuples of S. If the number of matching S tuples for each R tuple is small, index lookup is much more efficient than scanning the whole S relation. However, index lookup usually involves random I/Os. If the number of matching S tuples is large, it might be worthwhile considering block nested loops join.

Sort-Merge Join

If the join predicate is an equality predicate, sort-merge join can be used. The basic idea of the sort-merge join algorithm is to *sort* both relations on the join attribute and to then *merge* the sorted relations by scanning them sequentially and looking for qualifying tuples.

The sorting step groups all tuples with the same value in the join attribute together. Such groups are sorted based on the value in the join attribute so that it is easy to locate groups from the two relations with the same attribute value. Sorting operation can be fairly expensive. If the size of the relation is larger than the available memory, external sorting algorithm

```
Algorithm 2: Sort-Merge Join: R⋈R.r=S.s S
// sorting step
Sort the relation R on the attribute r;
Sort the relation S on the attribute s;
// merging step
R = first tuple in R;
S = first tuple in S;
S′ = first tuple in S;
while R ≠ eof and S′ ≠ eof do
    while R.r < S′.s do
        R = next tuple in R after R
    end
    while R.r > S′.s do
        S′ = next tuple in S after S′
    end
    S = S′;
    while R.r == S′.s do
        S = S′
        while R.r == S.s do
            add {R,S} to result;
            S = next tuple in S after S;
        end
        R = next tuple in R after R;
     end
    S′ = S;
end
```

is required. However, if one input relation is already clustered (sorted) on the join attribute, sorting can be completely avoided. That is why sort-merge join looks even more attractive if any of the input relations is sorted on the join attribute.

The merging step starts with scanning the relations R and S and looking for matching groups from the two relations with the same attribute value. The two scans start at the first tuple in each relation. The algorithm advances the scan of R as long as the current R tuple has an attribute value which is less than that of the current S tuple. Similarly, the algorithm advances the scan of S as long as the current S tuple has an attribute value which is less than that of the current R tuple. The algorithm alternates between such advances until an R tuple $\mathcal{R}$ and an S tuple $\mathcal{S}$ with $\mathcal{R}.r = \mathcal{S}.s$. The join tuple $\{\mathcal{R}, \mathcal{S}\}$ is added to result.

In fact, there could be several R tuples and several S tuples with the same attribute value as the current tuples $\mathcal{R}$ and $\mathcal{S}$. That is, several R tuples may belong to the current $\mathcal{R}$ group since they all have the same attribute value. The same applies to the current $\mathcal{S}$ group. Every tuple in the current $\mathcal{R}$ group joins with every tuple in the current $\mathcal{S}$ group. The algorithm them resumes scanning R and S, beginning with the first tuples that follow the group of tuples that are just processed.

When the two relations are too large to be held in available memory, one improvement is to combine the merging step of external sorting with the merging step of the join if the number of buffers available is larger than the total number of sorted runs for both R and S. The idea is to allocate one buffer page for each run of R and one for each run of S. The algorithm merges the runs of R, merges the runs of S, and joins (merges) the resulting R and S streams as they are generated.

If any of the two relations has an index on the join attribute, another improvement is to merge join the index instead of the relation. Sorting, in this case, can be avoided. After the join, it may be necessary to retrieve other attributes from the relation following qualifying index entries.

Hash Join

The hash join algorithm is commonly used in database systems to implement equijoins efficiently. In its simplest form, the algorithm consists of two phases: the "build" phase and the "probe" phase. In the "build" phase, the algorithm builds a hash table on the smaller relation, say R. In the "probe" phase, the algorithm probes the hash table using tuples of the larger relation, say S, to find matches.

The algorithm is simple, but it requires that the smaller join relation fits into memory. If there is not enough memory to hold all the tuples in R, an additional "partition" phase is required. There are several variants of the basic hash join algorithm. They differ in terms of utilizing memory and handling overflow.

Grace Hash Join The idea behind grace hash join is to hash partition both relations on the join attribute, using the *same* hash function. As the result, each relation is hashed into k partitions, and these partitions are written to disk. The key observation is that R tuples in partition i can join only with S tuples in the same partition i. If any given partition of R can be held in memory, the algorithm can read in and build a hash table on the partition of R, and then probe the hash table using tuples of the corresponding partition of S for matches.

If one or more of the partitions still does not fit into the available memory (for instance, due to data skewness), the algorithm is recursively applied. An

additional orthogonal hash function is chosen to hash the large partition into sub-partitions, which are then processed as before.

Hybrid Hash Join The hybrid hash join algorithm is a refinement of the grace hash join algorithm which takes advantage of more available memory. To partition R (S) into k partitions, the grace hash join uses one input buffer for reading in the relation and k output buffers, one for each partitions.

Suppose there is sufficient memory to hold an in-memory hash table for one partition, say the first partition, of R, the hybrid hash join does not write the partition to disk. Instead, it builds an in-memory hash table for the first partition of R during the "partition" phase. Similarly, when partitioning S, for the tuples in the first partition of S, the algorithm directly probes the in-memory hash table and writes out the results. At the end of the "partition" phase, the algorithm completes the join between the first partitions of R and S while partitioning the two relations. The algorithm then joins the remaining partitions as the grace hash join algorithm.

Compared with the grace hash join algorithm, the hybrid hash join algorithm avoids writing the first partitions of R and S to disk during the "partition" phase and reading them in again during the "build" and the "probe" phases.

Algorithm 3: Grace Hash Join: $R \bowtie_{r=s} S$
//partition R into k partitions
foreach $\mathcal{R} \in R$ **do**
  read $\mathcal{R}$ and add it to buffer page $h_1(\mathcal{R})$;
  flush the page to disk it full;
end
//partition S into k partitions
foreach $S \in S$ **do**
  read S and add it to buffer page $h_1(S)$;
  flush the page to disk it full;
end
//"build" and "probe" phases
for $i \leftarrow 1$ **to** k **do**
  foreach $\mathcal{R} \in$ *partition* R_i **do**
    read $\mathcal{R}$ and insert into the hash table using $h_2(\mathcal{R})$;
  end
  foreach $S \in$ *partition* S_i **do**
    read S and probe the hash table using $h_2(S)$;
    for matching $\mathcal{R}$ tuples, add $\{\mathcal{R},S\}$ to result;
  end
  clear the hash table and release the memory;
end

Aggregation

Simple aggregation operations include `MIN`, `MAX`, `SUM`, `COUNT`, and `AVG`. Aggregation operations can also be used in combination with a `GROUP BY` clause. See Fig. 4. `DISTINCT` clause that removes duplicate values can be viewed as a special aggregation operation too, which groups the input by values and outputs one row per group.

For aggregation queries without a `GROUP BY` clause, the algorithm is straightforward: scan the entire relation and update some state information for each scanned tuple. The state information varies for different aggregation functions. For `SUM`, the algorithm keeps track of the sum of the values retrieved so far. For `AVG`, the algorithm keeps track of the sum of the values and the total count of rows retrieved so far. For `COUNT`, the algorithm keeps track of the total count of values retrieved so far. For `MAX` (`MIN`), the algorithm keeps track of the largest (smallest) value retrieved so far.

For aggregation queries with a `GROUP BY` clause, the aggregation algorithms use either sorting or hashing.

Aggregation Based on Sorting

The sorting algorithm starts with sorting the relation on the grouping attributes and then scanning it again to compute the aggregations functions for each group. By sorting, the algorithm groups together all tuples with the same grouping attributes (therefore belong to the same group). The computation of the aggregation functions is similar to the way of aggregation computation without grouping, except that here the algorithm has to watch for group boundaries. When the algorithm sees a tuple with different grouping attributes, the current group finishes and a new group starts.

Aggregation Based on Hashing

The hashing algorithm first builds an in-memory hash table on the grouping attributes as it scans the relation. Each hash entry corresponds to a group and contains the

```
SELECT SUM (R. x)
FROM R
```

(1)

```
SELECT SUM (R. x)
FROM R
GROUP NY R.r
```

(2)

Evaluation of Relational Operators. Figure 4. Aggregation queries.

state information required by each aggregation function for the group. For each tuple, the algorithm probes the hash table to find the entry for the group to which this tuple belongs and update the state information. If such entry does not exist, the algorithm creates a new hash entry and initiates the state information.

If the relation is so large that the hash table does not fit in memory, the algorithm hash partitions the relation on the grouping attributes. Since all tuples in a given group are in the same partition, the algorithm can then scan each partition independently and compute aggregation functions as described before.

It is also possible and sometimes preferable to compute aggregation using an index as long as the index covers all the attributes required by the aggregation query. The advantage is to read in a much smaller index compared to the entire relation. If the grouping attributes form a prefix of the indexed keys, the algorithm can avoid the sorting step and scan the index entries sequentially.

Key Applications

Each database query is composed of a few relational operators. The final query plan may implement each relation operator in different ways to achieve an optimal performance. All the implementation techniques are widely used in database systems.

Cross-references

- ▶ Access Methods
- ▶ Buffer Pool
- ▶ Concurrency Control
- ▶ Cost Model
- ▶ External Sorting
- ▶ Hashing
- ▶ Parallel Query Processing
- ▶ Query Optimization

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.
3. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2002.
4. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Evaluation of XML Retrieval Effectiveness

▶ Evaluation Metrics for Structured Text Retrieval

Event

▶ Time Instant

Event and Pattern Detection over Streams

Mingsheng Hong, Alan Demers, Johannes Gehrke, Mirek Riedewald
Cornell University, Ithaca, NY, USA

Synonyms

Complex event processing (CEP); Event stream processing (ESP)

Definition

An event is a basic unit of information in streaming data. An event pattern is a combination of events correlated over time. Event pattern detection is an important activity in complex event processing. In this setting, the matches to the event patterns are referred to as complex events.

Historical Background

In the early 1990s, a set of pioneering work in *event systems*, such as SNOOP [3] and ODE [8], set out to define query languages for expressing event patterns. In these proposals, the data model for expressing events is not fixed. More recently, the approaches proposed by Cayuga [1,5,6] and SASE [14] for event pattern detection align more closely to relational query processing, in that each event is modeled by a relational schema, and some of the operators for expressing event pattern queries are drawn from relational algebra. Regardless of the data model for events, these systems all use some variant of NFA (Nondeterministic Finite state Automaton) as the processing model.

With the advent of internet-scale message brokering systems, *content based publish-subscribe systems*

such as [7] emerged. They are characterized by very limited query languages, allowing simple selection predicates applied to individual events in a data stream. Such systems trade expressiveness for performance – when well engineered, they exhibit very high scalability in both the number of queries and the stream rate. However, their inability to express queries that span multiple input events makes them unsuitable for event pattern detection.

Another category of systems closely related to event pattern detection is *stream databases*, such as Aurora [2], STREAM [10], and TelegraphCQ [4]. Unlike content based publish-subscribe systems, they focus on expressiveness. Such systems have very powerful query languages, typically including a rich functionality and extending SQL with provisions for sliding window and grouping features. Though powerful, stream databases are not designed for detecting event patterns, and their query languages can be awkward for expressing event pattern queries. Moreover, there is little work on scaling up stream database engines with the number of concurrent queries that are reasonably sophisticated.

Foundations

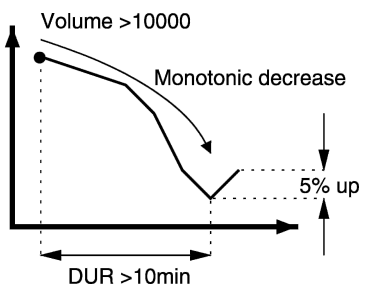
Event Pattern Query Model

An event stream is a potentially infinite sequence of events. Each event has a timestamp value, and its payload content is encoded by a relational tuple. Events are processed in timestamp order by an event processing system. They can be filtered, transformed, and correlated to form event patterns.

The computational model of matching event patterns over event streams naturally extends that of matching regular expression patterns over character streams in the following aspects. First, each stream event can contain multiple attribute-value pairs conforming to a relational schema, where the value domains are potentially infinite. In comparison, each character in a character stream provides for the same single attribute a value drawn from a finite alphabet. Second, given the multiple attributes for each stream event, event patterns can perform relational operations on the events, including filtering, projection and renaming of attributes. This provides expressive power especially to event correlation based on sequencing, where the attribute values of multiple stream events can be compared. Finally, event pattern detection involves a temporal aspect, in that stream events have timestamps, which event patterns reason about. In comparison, regular expression processing only involves character orderings in the streams, which can be viewed as a weaker notion of time.

Event patterns are usually expressed in an event algebra. Many such algebras have been proposed. One representative is the Cayuga algebra [5]. The Cayuga algebra is specifically designed for large-scale event pattern detection. In addition to unary operators for selection predicates and aggregates, the expressive power of this algebra comes from two binary operators. The first binary operator, *sequence*, can correlate two input events based on a join predicate. This join predicate involves timestamps, and can optionally involve other attributes in the stream schema. The second operator, *iteration*, is a generalization of the sequence operator. It is able to produce an event pattern involving arbitrarily many input events by iteratively concatenating input events with the pattern built so far. To allow users to interact with the system in a user-friendly way, a SQL-style query language, referred to as Cayuga Event Language (CEL) [6], has been developed. Figure 1 shows an event pattern query expressed in CEL. This event pattern query

```
SELECT Name, MaxPrice, MinPrice, Price AS FinalPrice
FROM
  FILTER{DUR > 10min}(
    (SELECT Name, Price_1 AS MaxPrice, Price AS MinPrice
    FROM FILTER{Volume > 10000}(Stock))
      FOLD{$2.Name = $.Name, $2.Price < $.Price}
    Stock)
    NEXT{$2.Name = $1.Name AND $2.Price > 1.05*$1.MinPrice}
  Stock
```



Event and Pattern Detection over Streams. Figure 1. Event pattern query to find stock price pattern.

corresponds to a particular trend in stock prices. Intuitively, this query searches for the given price pattern for any company. The pattern starts with a large trade (`Volume > 10,000`), followed by a monotonic decrease in price (`FOLD` clause), which lasts for at least 10 min (`DUR > 10min`). Then the price rebounds with a sudden increase by 5% (`NEXT` clause). The `NEXT` and `FOLD` constructs respectively correspond to the two binary operators in the Cayuga algebra introduced above. `NEXT` matches the next event in the stream that satisfies a given condition (same company name and 5% higher price in the example). `FOLD` is the iterated version of `NEXT`, i.e., it continues matching the next event that satisfies a certain property until a stopping condition is satisfied.

SASE [14] uses a query algebra similar to the Cayuga algebra, where sequence and iteration are the key primitives. SASE also supports *negation*-style event patterns, where a pattern is matched by the absence of an input event, rather than the presence. In addition to online event stream processing supported by Cayuga and SASE, work on sequence database systems has focused on matching event patterns offline over archived data [11,12,13]. Sequence is again the key primitive for expressing event patterns in the SEQ query algebra [13] and the SQL-TS query language [12].

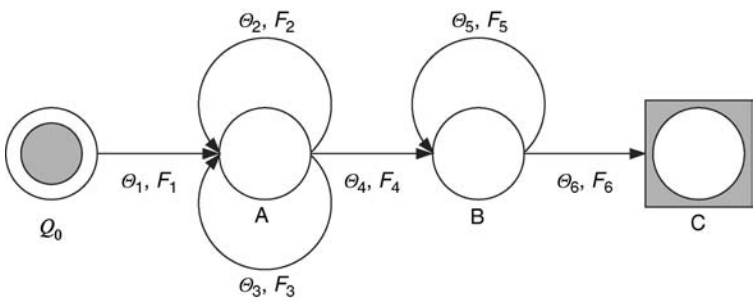
Event Pattern Query Processing

As is mentioned earlier, an event pattern query is typically implemented by a state machine. Continuing the above query example, this approach is described in the context of Cayuga. Each Cayuga automaton is an extension to the classical non-deterministic finite automaton [9] in the following aspects. First, each automaton edge is associated with a predicate, and for an incoming event, this edge is traversed if and only if the predicate is satisfied by this event. This mechanism implements the selection predicates in the event patterns. Second, when patterns are matched, to be able to generate witness events with concrete content instead of boolean answers, each automaton instance needs to store the attributes and values of those events that have contributed to the pattern instance.

Figure 2 shows the automaton for the example query in Fig. 1. The two middle states correspond to the `FOLD` and `NEXT` operators, respectively. The predicates θ_i associated with automaton edges originate either from `FILTER` conditions (θ_1, θ_6 in the example) or from join conditions of the `FOLD` and `NEXT` operators. Specifically, θ_1 implements the filter predicate `vol` > 10,000. When an input stock event e satisfying θ_1 occurs, a new automaton instance I is created under state A, remembering the content of e.

Each automaton instance encodes a particular event pattern built from the prefix of the event stream that has been processed so far. When an automaton instance reaches the final state C, a match to the entire event pattern specified in the query has been found, and will be output by this automaton.

To continue the explanation of the example Cayuga automaton in Fig. 2, θ_2 and θ_3 respectively implement the two join predicates associated with `FOLD` in the query shown in Fig. 1. These two predicates on the two self-loop edges associated with state A together build a monotonically decreasing sequence in the prices of a particular stock. Specifically, after the occurrence of event e, when a later stock event e' together with I satisfies θ_2; i.e., e' and I have the same stock name s (say), θ_3 is evaluated to check whether this event pattern built so far can be extended. For this reason, θ_2 serves as a criterion which, when satisfied, concatenates the next input event from `Stock` to the event pattern built up so far, and θ_3 serves as a criterion



Event and Pattern Detection over Streams. Figure 2. Cayuga automaton example (source: [6]).

which, when satisfied, continues the extension of the event pattern being built. The event building process proceeds similarly to state B and C. The functions F_i are responsible for transforming event stream schemas and automaton state schemas.

Event pattern queries expressed in SASE and SQL-TS are processed in a similar way. In SASE, NFA is one of the run-time operators. Operator re-ordering is performed as part of the query optimization to produce efficient query plans. For example, a selection predicate above an NFA operator can be pushed inside the NFA operator to discard irrelevant input events earlier, thus improving system throughput. In SQL-TS, to optimize pattern search, the query engine exploits the inter-dependencies between the elements of a sequential pattern.

Evaluating one event pattern query efficiently is relatively easy. However, it is challenging to efficiently evaluate a large number of concurrent event patterns. Multi-Query Optimization (MQO) techniques have been developed to share the computation among concurrent event patterns. For example, in Cayuga, the event processing engine achieves this goal by exploiting the relationship of the query algebra to the automata-based query execution, and the commonality among queries. Specifically, each query is first translated into a set of automata. These automata are then "merged" with existing ones in the engine. During the merging process, two optimization techniques are used. First, two automata with the same prefix of states can merge these states, thus sharing computation and storage. This is similar to the technique of finding common subexpressions in relational query processing. Second, some of the filtering predicates on the automaton edges can be managed efficiently by indexes in a way similar to techniques for processing multiple selection operators [7]. These two techniques enable a throughput of thousands of events per second, even for tens of thousands of active event pattern queries.

Key Applications

Event pattern detection targets a large class of both well-established and emerging applications, including supply chain management for RFID (Radio Frequency Identification) tagged products, real-time stock trading, monitoring of large computing systems to detect malfunctioning or attacks, and monitoring of sensor networks, e.g., for surveillance. These *event monitoring applications* need to process massive streams of events in (near) real-time. There is great interest in these applications as indicated by the establishment of sites like http://www.complexevents.com, which bring together major industrial players like BEA, IBM, Oracle, and TIBCO.

Future Directions

One important future direction is to integrate event processing systems with stream databases. In the past they have evolved along different paths, and are designed for different query workloads, as is described in the Historical Background section. It would be beneficial to integrate these two categories of stream processing systems, for two reasons. First, there is much overlap in their functionality. For example, both categories support stateless operations such as filtering and projection, and some forms of stream join. Second, there is a class of stream applications that demand functionality from both categories. There are two major challenges in the integration. At the logical level, a unified query algebra is needed for expressing queries in both categories. At the physical level, it is desirable to evaluate both categories of queries in the same engine. This is a challenging task since current stream database engines, like relational database engines, are usually based on operator trees, while event engines are based on variants of NFAs.

Cross-references

- ▶ Complex Event Processing
- ▶ Continuous Query
- ▶ Data Stream Management Architectures and Prototypes
- ▶ Publish/Subscribe over Streams
- ▶ Stream Processing

Recommended Reading

1. Brenna L., Demers A., Gehrke J., Hong M., Ossher J., Panda B., Riedewald M., Thatte M., and White W. Cayuga: a high-performance event processing engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1100–1102.
2. Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring streams – a new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 215–226.
3. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite events for active databases: semantics, contexts and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
4. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S.R., Raman V., Reiss F., and Shah M.A. Telegraph CQ: continuous

dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
5. Demers A., Gehrke J., Hong M., Riedewald M., and White W. Towards expressive publish/subscribe systems. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 627–644.
6. Demers A., Gehrke J., Panda B., Riedewald M., Sharma V., and White W. Cayuga: a general purpose event monitoring system. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 412–422.
7. Fabret F., Jacobsen H.A., Llirbat F., Pereira J., Ross K.A., and Shasha D. Filtering algorithms and implementation for very fast publish/subscribe. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 115–126.
8. Gehani N.H., Jagadish H.V., and Shmueli O. Composite event specification in active databases: model and implementation. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 327–338.
9. Hopcroft J.E., Motwani R., and Ullman J.D. Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, MA, USA, 2nd ed., 2000.
10. Motwani R., Widom J., Arasu A., Babcock B., Babu S., Datar M., Manku G.S., Olston C., Rosenstein J., and Varma R. Query processing, approximation, and resource management in a data stream management system. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
11. Ramakrishnan R., Donjerkovic D., Ranganathan A., Beyer K.S., and Krishnaprasad M. SRQL: sorted relational query language. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management, 1998, pp. 84–95.
12. Sadri R., Zaniolo C., Zarkesh A.M., and Adibi J. Optimization of sequence queries in database systems. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 71–81.
13. Seshadri P., Livny M., and Ramakrishnan R. SEQ: a model for sequence databases. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 232–239.
14. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.

Event Broker

► Request Broker

Event Causality

Guy Sharon
IBM Research Labs-Haifa, Haifa, Israel

Definition

The definition of an event processing network includes a relation to express un-modeled event processing logic between events.

The fact that events of type *eT1* cause events of type *eT2* is denoted by the relation *causes(eT1,eT2)*.

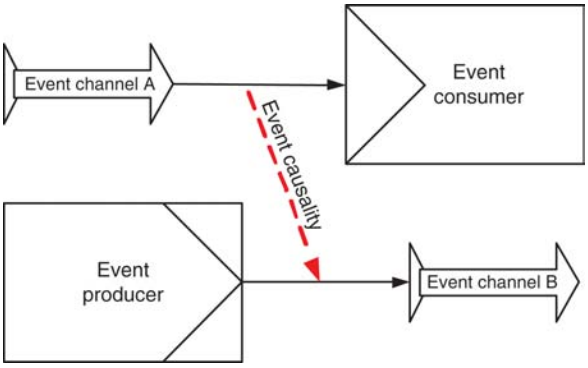
In an EPN where E is the set of edges representing event streams, EC is the set of Event Channels, C is the set of Event Consumers and P is the set of Event Producers, the relation is evaluated to be true if events of type *eT1* flow in an event stream $e1(u,v)$: $e1 \in E$, $u \in EC$, $v \in C$ and events of type *eT2* flow in an event stream $e2(m,l)$: $e2 \in E$, $m \in P$, $l \in EC$ and there is some un-modeled event processing logic between events of type *eT1* consumed by $v \in e1$ and events of type *eT2* produced by $m \in e2$.

Key Points

The event processing intent defined by an event processing network may not cover the entire flow of events through systems as there may be cases where an event is handled by an event consumer, such as a software application, and as a result the application publishes, as an event producer, a new event to an event processing network. In essence, there is some un-modeled processing logic performed by the application that results in a new event that may be processed further on. The specifics of this logic are not important for the realization of an event processing network as this logic is implemented and executed by a consumer such as an application, however, having the relation *causes(eT1,eT2)* as part of the model enables to describe the complete event-driven interactions in systems and is used in performing model analysis such as termination analysis [1–3].

In the model, event causality is represented as a red, dashed, and directed edge from the event stream being consumed to the event stream being produced (Fig. 1).

The definition of event causality in an event processing network is meant to be theory neutral and



Event Causality. Figure 1. Event causality.

corresponds to common sense causal interpretation in the physical world. Applying the many theories about causation [4,5] to the cause of an event by another can extend the perception of the event-driven interactions and perhaps most productively enhance analysis capabilities of models, but for the purpose of the realization of an event processing network this is not necessary.

Cross-references

► Event Processing Network

Recommended Reading

1. Aiken A., Hellerstein J.M., and Widom J. Static analysis techniques for predicting the behavior of active database rules. ACM Trans. Database Syst., 20(1):3–41, 1995.
2. Bailey J. and Poulovassilis A. Abstract interpretation for termination analysis in functional active databases, J. Intelligent Inf. Syst., 12(2–3):243–273, 1999.
3. Baralis E., Ceri S., and Paraboschi S. Compile-time and runtime analysis of active behaviors, IEEE Trans. Knowledge Data Eng., 10(3):353–370, 1998.
4. Fisk M. A defence of the principle of event causality. Br. J. Philos. Sci., 18(2):89–108, 1967.
5. Pearl J. Causality: Models, Reasoning, and Inference. New York: Cambridge University Press, 2000.

Event Causality Graph

► Event Flow

Event Channel

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event connection; Event pathway; Event topic

Definition

An event channel is a routing node in the event processing network that receives events from event sources and event processing agents, and route them to event sinks and event processing agents.

Key Points

An event channel [2] is the main routing vehicle in the event processing network; it is being used to enable loosely coupled architecture, in which the event producer and event consumer does not have any dependency among them [1]. An event channel receives an event (in push or pull) from an event source or from an event processing agent, and makes routing decisions. The routing decision may be of several types:

1. Itinerary based routing: the consumers are explicitly specified within the event content.
2. Subscription based routing: using a "callback" protocol, the consumer subscribes to events on the channel.
3. Intelligent routing: a decision process is activated to decide on the event consumers.
4. Calendar based routing: Routing is controlled by a calendar function.

The routing may be partitioned according to context [3], if context is supported.

Cross-references

► Channel-Based Publish/Subscribe
► Context
► Event Processing Agent
► Event Processing Network
► Event Sink
► Event Source
► Event Stream

Recommended Reading

1. Crowcroft, J. Channel Islands in a Reflective Ocean: Large Scale Event Distribution in Heterogeneous networks. In Proc. 2nd Int. IFIP-TC6 Networking Conf., 2002, pp. 1–9.
2. Luckham, D., Schutle, R. (eds). EPTS Event Processing Glossary version 1.1 http://complexevents.com/?p = 409
3. Sharon, G. Etzion, O. Event processing network: Model and implementation. IBM System Journal, 47(2):321–334, 2008.

Event Cloud

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Definition

Event Cloud is a partially ordered set of events (poset), either bounded or unbounded, where the partial orderings are imposed by the causal, timing and other relationships between the events [2].

Key Points

Event cloud [1] consists of a set of events and one or more partial order relations. The following event relations are being used:

- Causality relation: Each collection of events can be considered as an *event cloud*, the causality relation creates a partial order relation among a collection of events, causality is transitive, anti-symmetric, and non-reflexive.
- Precedes relation: defined as event e1 precedes event e2, if it occurs in e1 occurs in reality before event e2. This is a partially order set – since the relative timing may not be known, or may overlap, if occur during an interval.

Event cloud can be defined over the entire collection of events that flow through some system within a certain bounded or unbounded time[3,4]. A special case of event cloud is all events that are input to a single event processing agent and are ordered as a time-series, in this case the event cloud is a collection of event streams.

Cross-references

► Complex Event Processing
► Event Stream

Recommended Reading

1. Luckham, D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 2002.
2. Luckham, D., Shulte, R. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409
3. Rozsnyai, S., Vecera, R., Schiefer, J., and Schatten, A. Event cloud – searching for correlated business events. In Proc. 9th IEEE Int. Conf. on E-Commerce Technology & 4th IEEE Int. Conf. on Enterprise Comp., E-Commerce and E-Services, 2007, pp. 409–420.
4. Widder, A., von Ammon R., Schaeffer, P., and Wolff, C. Identification of suspicious, unknown event patterns in an event cloud. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, pp. 164–170.

Event Composition

► Event Detection

Event Composition (Partial Overlap)

► Event Pattern Detection

Event Connection

► Event Channel

Event Consumer

► Event Sink

Event Control

► Event Detection

Event Declaration

► Event Specification

Event Definition

► Event Specification

Event Detection

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

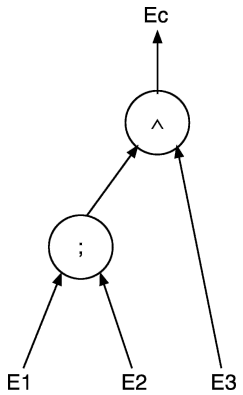
Chronicle recognition; Event trace analysis; Event control; Event composition; Monitoring of real-time logic expressions

Definition

Event detection is the process of analyzing event streams in order to discover sets of events matching patterns of events in an event context. The event patterns and the event contexts define event types. If a set of events matching the pattern of an event type is discovered during the analysis, then subscribers of the event type should be signaled. The analysis typically entails filtering and aggregation of events.

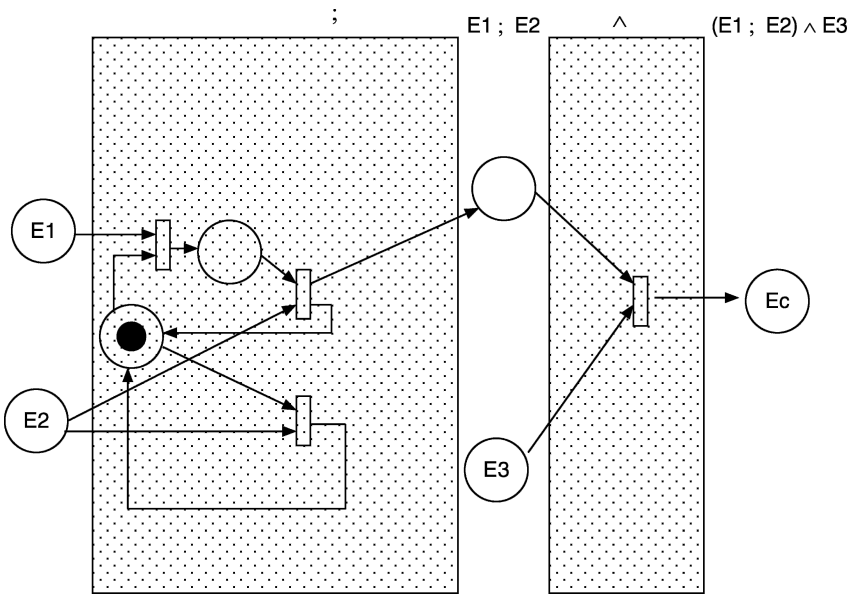
Historical Background

Seminal work on event detection was done in HiPAC [7,11] and Snoop [8,9] as well as in ODE [14] and SAMOS [13]. Essentially, in Snoop, ODE, and SAMOS different methods for realizing the matching of event detection were investigated. In Snoop, implementations of the event operators are structured according to the syntax tree of the event expression, where each node represents an event operator. The event operator in a node is the principal event operator of the event type it corresponds to and each event operator is implemented as a subprogram (e.g., procedure, method). The procedures are invoked during evaluation of the expression. In contrast, in SAMOS [13] operators expressed as Petri-nets are combined; and in ODE [14], finite state automatas built from the event specifications are employed.

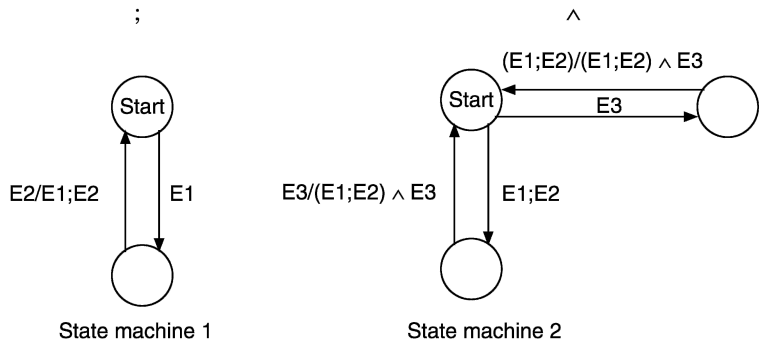


Event Detection. Figure 1. Syntax tree of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.6].

For example, consider the expression E_c **defined by** $(E_1;E_2) \wedge E_3$ that means an E_c occurs if a sequence of E_1 and E_2 and an event of type E_3 occurs. This can be transformed into the syntax tree in Snoop as depicted in Fig. 1, since the conjunction '$\wedge$' is the principal event operator of E_c. As in data flow machines, the idea is that the nodes are evaluated according to some policy; for example, a common policy is that as soon as there are data available (events) on the input edges to an event, the operator is evaluated. Thus, the node ';' is evaluated if there are E_1, E_2 or both E_1 and E_2 events available. In colored Petri-nets used in SAMOS, the expression E_c is represented as depicted in Fig. 2. The events are represented as tokens in the places. Further, the event expression can be translated into a finite state automata as in ODE (depicted in Fig. 3), where the transitions are labeled with *Event/Action* where actions denoted E_x means generate a composite event occurrence E_x. A significant difference between the three different representations of event operators is whether they are capable of handling multiple situations in which events are generated concurrently. The approaches in Snoop and SAMOS can handle concurrent situations, whereas finite state automatas must be augmented with processes executing the finite state



Event Detection. Figure 2. Petri-net representation of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.7].



Event Detection. Figure 3. Finite state automata representation of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.8].

automatas to enable this. The rest of the historical background address the following significant issues: *(i)* management of the combinatorial explosion in the analysis of event streams; *(ii)* parameterized event detection, and *(iii)* design issues.

In Snoop [8,9], event contexts or event parameter consumption modes were introduced handle the combinatorial explosion, issue *(i)*, in a systematic way. Each event context represents some semantic or pragmatic knowledge about the situations that are monitored. For example, in recent event context only the most recent event occurrences are allowed to be used to form composite events. This context is meant to be used in, for example, real-time applications. In contrast, in chronicle event context event matching is based on historical order and is meant to be used to monitor, for example, that communication protocols are not violated.

Another way to handle the combinatorial explosion is to limit the size of the past. Essentially, this is a technique from distributed, parallel, and real-time systems, in which time intervals are used to determine the validity of events. For example, in remote procedure calls timeouts are generally used to detect failures such as server node crashes or network packet omissions – timeouts occur if there is no response within a specified time interval. The timeouts can be used to limit the past by removing events that can no longer be part of a composite event in the future. This technique has been suggested in related areas, for example, chronicle recognition [12], monitoring of real-time logic [10], and general monitoring of distributed systems [17]. For example, the technique of using time windows to limit processing has been successfully employed in event detection in real-time control [19].

One significant issue is parameterized event detection (issue *(ii)*), because it is often the case that events in event specifications must carry parameters denoting the situation in which the events were generated. In SAMOS, a limited form of parameterized event detection is allowed by either requiring that events stem from the same transaction or the same user. Bækgaard and Godskesen [2] introduced specification of parameterized events based on TCCS (temporal calculi for communicating systems). However, they have not shown how to transform the specification into code or configuration of event detection. In EPL [20] and XChange^{EQ} [4] parameterized event detection are addressed. Both EPL and XChange^{EQ} stem from databases, for example, an incremental join algorithm for parameterized event detection is proposed by Bry and Eckert [5].

Significant results concerning design issues, issue *(iii)*, are that the following features are among the most important: storage structures for event streams, indexing techniques of event streams, handling of isolation property of transaction with respect to event detection as well as memory management techniques. The BEAST benchmark [15] concluded that indexing techniques and storage structures are more important than, for example, choosing Petri-nets, finite state automata or code to perform the matching. In DeeDS [1], the isolation property is violated in a controlled way by integrating event detection in the transaction processing in such a way that events can violate isolation if explicitly allowed to. The events are cached in a separate filtered event log as well as stored as first class objects in the database. This entails that the problem of doing event detection in the scope of transaction is removed, that is, event objects are not locked as is the case when event detection is performed within transactions.

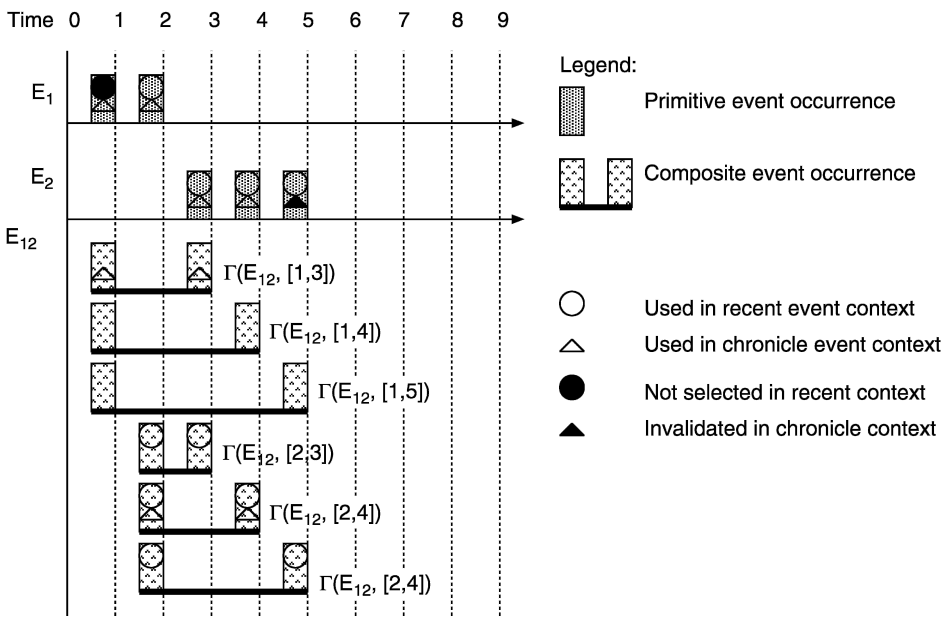
The drawback is that the filtered event log must be maintained, that is, support recovery processing and the isolation property as well as support pruning of invalid events.

Foundations

There are several issues that are significant. These issues can be categorized into a usage and a development perspective. The use of event detection entails two difficult problems: *(i)* understanding what is actually happening and *(ii)* dealing with event detection in different temporal scopes. The first problem has been considered in, for example, visualization of event detection [3]. It was observed that master's students had difficulty in understanding event detection when the topic was covered in graduate courses. The most likely explanation is that, in contrast to condition evaluation, it is also necessary to know the previous results of event detection to determine the result of the current evaluation. For example, consider Fig. 4 for E_{12} **defined by** $E_1;E_2$ where $\Gamma(E,[t_1,t_2])$ reads that "an event of type E has occurred throughout the interval $[t_1,t_2]$"; in this example, the different results of event detection can be confusing. All variations of E_{12} are discovered in general (unconstrained) event context, and the composite events marked with circles are discovered in the recent event context, whereas the composite events marked with triangles are discovered in chronicle event context. Note that, for example, to understand that $\Gamma(E,[2,4])$ has been discovered require knowledge of that $\Gamma(E[1,3])$ has been discovered and the consequences for future event detection. Thus, there is need for tools to help developing, verifying and validating event specifications. These tools require that the event detection process is instrumented in such a way that internal states can be observed.

The second problem relates to parameterized event detection. This is typically used to enable matching of events with the same temporal scope (e.g., same transaction, same process, accessing the same object). One problem is how these temporal scopes relate to each other, for example, a transaction can encompass multiple processes and these processes can serve several transactions. Sometimes it is necessary to mix temporal scopes, for example, the occurrence of an event outside a temporal scope may negate the occurrence of a composite event within the scope.

From a development perspective of ADBMS and event detection in particular, there are two issues of importance: *(i)* data structures for efficient event detection, and *(ii)* decentralized event detection. In database prototypes simple queue structures has been employed to store events, whereas, for example, some kind of a heap structure may provide better results. In particular, if event detection is explicitly invoked rather than implicitly invoked each time an event occurs, then



Event Detection. Figure 4. Example of event detection in some event contexts [18, Fig. 6.1].

more advanced data structures can improve the performance and time complexity of event detection. Concerning issue *(ii)*, event detection is suitable for decentralized processing, since it follows the data flow machine idea. However, there are a lot of issues that must be considered in such situations. Guiding requirements are, among others, interoperability requirements and the filtering effect of event detection for a particular domain. An example of the latter requirement is network event correlation [16], in which a major part of network events filtered out and the design of event detection should be optimized for pruning events.

In terms of performance, the algorithmic time complexity depends on a number of factors: the type of event context and whether a single signaled event is evaluated (as in implicit invocation of event monitoring) or multiple signaled events are evaluated (as in explicit invocation of event monitoring). In general, the difference in terms of algorithmic time complexity between different event contexts disappears if multiple events are evaluated. Table 1 describes the algorithmic time complexity in situations where all signaled events results in composite events. The meaning of the symbols are as follows: H is the event history (set of buffered events), M is the set of event types, E is an event expression, and P is a the set of parameters carried by each event. The function f depends on how event types are indexed in the system. For example, if the red-black tree technique is employed for indexing then $f(n) = \log(n)$, whereas $f(x) = 1$ if hashing is employed instead. Other event contexts (e.g., [6]) may have different algorithmic time complexity characteristics.

The most important factor is the size of the event history and by keeping this as short as possible it is even possible to employ the unconstrained event context in practice (cf. [19]). Event expressions are typically short unless they are automatically generated from some other specifications. For example, if a set of event types is required to occur in a particular order and it is necessary to detect failures to do so as soon as possible, then it is necessary to monitor all permutations of the correct order and in this case the size of event expressions can become uncommonly large. The size of the parameters is typically not a problem in high bandwidth systems and is less significant compared to the event history and the size of the event expressions. Finally, the number of event types in a system is insignificant as long as proper indexing is employed.

Event Detection. Table 1. Algorithmic time complexity of event detection [18].

Event context	Single	Multiple
Recent	$O(f(\|M\|) + \|E\|\|P\|)$	$O(\|H\|(f(\|M\|) + \|E\|\|P\|))$
Chronicle		
Continuous	$O(f(\|M\|) + \|H\|\|E\|\|P\|)$	
Cumulative		
General (unconstrained)	$O(\|H\|^{\|E\|})$	

Key Applications

The key users of this area are the researchers and developers of active functionality in, for example, active databases.

Future Directions

Some future issues in event detection is the following:

1. To develop tools and techniques for developing, understanding, verifying and validating event specifications. As addressed in this entry, the understanding of event detection is more complex than condition evaluation and this is a really crucial issue for successful use of event detection.
2. To investigate relevant data structures to improve event detection, in particular, parameterized event detection as well as decentralized event detection. If large event expressions are prevalent in a system, then this issue becomes critical.
3. To further investigate the performance characteristics in, for example, decentralized event detection. The reported study [18] has been performed for centralized event detection. Further work need to be done in other settings.
4. To investigate fault-tolerant event detection in various settings such as mobile ad-hoc networks.
5. To investigate if fuzzy reasoning inside event detection can be used. For example, in monitoring of real-world situations, it is necessary to handle uncertainties of various kinds.

Cross-references

► Active Database Execution Model
► Active Database Knowledge Model
► ADBMS (Active Database Management System) Architecture
► Atomic Event

- ▶ Composite Event
- ▶ Event
- ▶ Event specification

Recommended Reading

1. Andler S., Hansson J., Eriksson J., Mellin J., Berndtsson M., and Eftring B. DeeDS towards a distributed active and real-time database system. Special Issue on Real Time Data Base Systems. ACM SIGMOD Rec., 25(1):38–51, 1996.
2. Bækgaard L. and Godskesen J.C. Real-time event control in active databases. J. Syst. Softw., 42(3):263–271, 1997.
3. Berndtsson M., Mellin J., and Högberg U. Visualization of the composite event detection process. In Proc. Int. Workshop on User Interfaces to Data Intensive Systems, 1999.
4. Bry F. and Eckert M. Rule-based composite event queries: the language XChangeEQ and its semantics. In Proc. 1st Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 16–30.
5. Bry F. and Eckert M. Temporal order optimizations of incremental joins for composite event detection. In Proc. Inaugural Int. Conf. on Distributed Event-Based Systems, 2007, pp. 85–90.
6. Carlsson J. Event Pattern Detection for Embedded Systems. Ph.D. thesis no 44, Mälardalen University, 2007.
7. Chakravarthy S., Blaustein B., Buchmann A.P., Carey M., Dayal U., Goldhirsch D., Hsu M., Jauhuri R., Ladin R., Livny M., McCarthy D., McKee R., and Rosenthal A. HiPAC: a research project in active time-constrained database management. Tech. Rep. XAIT-89-02, Xerox Advanced Information Technology, 1989.
8. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite events for active database: semantics, contexts, and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
9. Chakravarthy S. and Mishra D. Snoop: an event specification language for active databases. Knowl. Data Eng., 14(1):1–26, 1994.
10. Chodrow S.E., Jahanian F., and Donner M. Run-time monitoring of real-time systems. In Proc. Real-Time Systems Symposium, 1991, pp. 74–83.
11. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC project: combining active databases and timing constraints. ACM SIGMOD Rec., 17 (1):51–70, 1988.
12. Dousson C., Gaborit P., and Ghallab M. Situation recognition: representation and algorithms. In Proc. 13th Int. Joint Conf. on AI, 1993, pp. 166–172.
13. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
14. Gehani N.H., Jagadish H.V., and Schmueli O. COMPOSE – a system for composite event specification and detection. In Advanced Database Concepts and Research Issues. Springer, Berlin, 1993.
15. Geppert A., Berndtsson M., Lieuwen D., and Roncancio C. Performance evaluation of object-oriented active database management systems using the BEAST benchmark. Theor. Pract. Object Syst. 4(4):1–16, 1998.
16. Liu G., Mok A., and Yang E. Composite events for network event correlation. In Proc. IFIP/IEEE Int. Symp. on Integrated Network Management, 1999.
17. Mansouri-Samani M. and Sloman M. GEM: a generalized event monitoring language for distributed systems. IEE/IOP/BCS Distrib. Syst. Eng. J., 4(2):96–108, 1997.
18. Mellin J. Resource-Predictable and Efficient Monitoring of Events. Ph.D. thesis no 876, University of Linköping, 2004.
19. Milne R., Nicol C., Ghallab M., Trave-massuyes L., Bousson K., Dousson C., Quevedo J., Martin J.A., and Guasch A. TIGER: real-time situation assessment of dynamic systems. Intell. Syst. Eng., 3(3):103–124, 1994.
20. Motakis I. and Zaniolo C. Composite temporal events in active database rules: a logic-oriented approach. In Proc. 4th Int. Conf. on Deductive and Object-Oriented Databases, 1995, pp. 19–37.

Event Driven Architecture

K. Mani Chandy
California Institute of Technology, Pasadena, CA, USA

Synonyms

Event driven service-oriented architecture; Event processing systems; Sense and respond systems; Sensor network systems; Active databases; Streaming database systems

Definitions

An event driven architecture is a software architecture for applications that detect and respond to events. An event is a significant change in the state of a system or its environment. The change may occur rapidly or slowly. The occurrence of an event in the past, or its current unfolding, or a prediction of an event in the future is deduced from data. An event-driven architecture includes sensors and other sources of data; processors that fuse data from multiple sensors and detect patterns over time, geographical locations, and other attributes and deduce events that occurred or predict events; responders for initiating actions in response to events; communication links for transferring information between components; and administrative software for monitoring, tailoring and managing the application.

Historical Background

Event driven systems have been used in military command and control systems, and in SCADA

(supervisory control and data acquisition) systems for decades. Platforms for event driven applications have been developed by several companies in the last decade; these general-purpose platforms can be tailored to obtain special-purpose event-driven applications. The development of event-driven platforms has evolved from several fields including databases, message-driven middleware, rule-based systems, business intelligence, statistics, and command and control systems.

Foundations

The metrics for evaluating event driven applications are different from those used to evaluate many other information technology systems. A discussion of the metrics helps highlight the goals of event driven systems and clarifies the differences between these systems and more traditional information technologies.

Measures for Evaluating Event Driven Applications

Event-driven applications detect and respond to events. Examples of such applications include those that: take action when earthquakes, tsunamis, or hurricanes are imminent; help manage investment portfolios by sensing risks and opportunities in markets; help disabled or aging people live independently by detecting and responding to potential problems; and prevent crime by identifying and responding to threats.

Measures by which event-driven applications are evaluated include the usual measures of total cost of ownership and measures, called the *REACTS* metrics (after the first letters of the measures), which are more specific to event driven architectures.

- *Relevance*: This measures the relevance of information generated by the system to consumers of the information. A message about a tsunami warning sent to a person's phone is highly relevant while she is on a beach where the tsunami is about to strike but is less relevant while she is hundreds of miles away. Relevance depends on the state of the receiver. Systems that send a great deal of irrelevant information (spam) are likely to be ignored.
- *Effort*: This measures the effort required by operational people to define specifications of events and responses and by IT staff to integrate event applications with existing systems. The time taken by end users, such as stock traders, to tune systems to meet their specific needs, and the time taken by IT staff to integrate event applications with existing infrastructure are major costs.
- *Accuracy*: This measures the costs of inaccurate information, such as false positives. A false tsunami warning gets beaches to be evacuated unnecessarily. High frequencies of inaccurate information result in information being ignored.
- *Completeness*: This measures the costs of incomplete information. For example, the absence of a hurricane warning can results in thousands of deaths. A single false negative has a higher cost than a single false positive in most situations; however, systems are often designed with very high rates of false positives compared to rates of false negatives.
- *Timeliness*: This measures the costs of detecting events too late. The benefit of detecting an event depends on the time available to execute an appropriate response. There is little added benefit in having millisecond responses when responses in minutes will do. There are, however, significant costs in responses initiated too late: a tsunami warning received after a tsunami strikes offers no value.
- *Security*: This measures the costs of reducing and managing intrusions and other attacks. Event driven applications often run for extended periods and manage critical infrastructure. For example, new infrastructure in the electrical power grid enables utilities to control airconditioners and other devices in homes and offices. Attacks that shut off power to homes and factories have severe consequences.

The design of event driven applications is a process of evaluating, comparing, and trading off REACTS measures for different design options.

Components of Event Driven Architectures

An event – a significant state change – is different from the data that describes the event; data that describes an event is often called an *event object*. Event objects may be stored in databases, sent as messages, or used within programs. An action, such as issuing a warning about a possible fraud, is itself an event. Information about the action is captured in an event object or is not captured at all.

Event driven architectures consist of data acquisition components such as sensors; event processors that integrate data from multiple sensors and databases; responders that initiate appropriate actions after

events are detected; communication links that transmit information between components and administrative services that help in specifying and managing the operation of event driven applications. Next, each of these components is discussed briefly.

Sensors: Data acquisition components acquire data from the environment and from within the enterprise; this data is analyzed to detect patterns that signify events. Data may be pushed to these components or the components may acquire data by polling services. For example, stock ticker feeds are data acquisition components that push stock ticker information to the application without having the application make an explicit request for each stock price update. Other data acquisition components poll Web sites or other services according to fixed schedules or when requested to do so by event processors. These components send the polled information – or the difference between data obtained on successive polls – to the application, or store the data for later processing. Polling too frequently is onerous for the site being polled, while polling infrequently can increase delays in event detection.

Event Processors: The role of event processors is to integrate data from multiple sensors, add value to event objects by incorporating data from databases and business intelligence repositories, find patterns that signal events, and initiate actions by responders. Event processors may also request sensors to obtain additional data, providing closed-loop coupling between data acquisition and processing. Common functions of event processors include the following:

- Event Filtering: Data may be generated by sensors, such as stock tickers or RFID readers, at rapid rates and only a fraction of this data may be relevant to an application. Event filters are specified by the data that they pass through to later stages of processing. Filters located near sensors, physically or logically, reduce the amount of communication.
- Event Object Enrichment: Enriching an event object by adding relevant data from databases helps in later stages of processing. For example, an event object describing a customer's request for a new service can be enriched by adding data about the customer's history.
- Event Fusion: Sequences of event objects sent by multiple sensors and event processors are integrated, or "fused" by other event processors, to generate new event objects. For example, streams of event objects generated by multiple package-delivery trucks are fused into a single stream dealing with all packages.
- Event Stream Splitting: A stream of event objects can be split into multiple streams each of which deals with a separate set of attributes; processing streams of event objects all of which deal with the same topics is simpler.
- Time Series Analysis: Event processors in many applications carry out time series analyses to detect anomalous behavior or to determine if behaviors have changed significantly over time.
- Complex Event Processing and Event Prediction: The detection of complex patterns requires analytics that spans time, geographical locations, and business functions. Initiating responses in preparation for predicted events reduces reaction delay. The detection of complex patterns in the historical record and the accurate prediction of future events often requires complex processing.

Responders

A typical response is the initiation of a business process. Example responses include updating dashboards that display key performance indicators, sending alerts, setting up temporary task forces to respond to an event, and updating databases and logs for later analysis. The execution of the business process responding to an event may, in turn, generate new events.

Communication Layer

Communication among sensors and event processors, in some applications, is by means of wireless. In some applications, wireless bandwidth may be low because sensors may be required to be inexpensive, power may have to be conserved, or the environment may be noisy. The geographical layout of components communicating by wireless is critical in such applications.

The logical architecture for communication among components in many applications is publish/subscribe. Some components publish event objects, and some components subscribe for specified types of event objects with specified ranges of parameters. The pub/sub network ensures delivery of event objects to appropriate destinations. The logical communication architecture can also be point-to-point, pushing event objects asynchronously from one point in the network to the next, or a broadcast system.

Administration Layer

The administration layer is used for several purposes including the following: initially to tailor a general-purpose event driven architecture to suit a specific application; at run time to monitor an application; and, at redesign time to replay scenarios for forensics and for adapting systems to changing conditions. Since event driven applications are often used to control physical infrastructures over long periods, monitoring sensors and responders is an essential aspect of these applications. Thus, the administrative layer of an event driven application is itself an event driven application.

System Specification

The process of implementing an event driven application is, in essence, an application integration activity: sensors, processors and responders have to be integrated into operations. The specification of an event driven application consists of specifications of the sensing, processing and responding subsystems and their integration. The subsystems are quite different from each other; therefore, the notations used in specifying them are different.

Specifications of sensing subsystems include identification of what data sources and sensors to use, where they should be placed, how they should communicate, activation and polling schedules, and mappings from data models used by the external environment to data models used within the application. Specifications of event processing subsystems deal with functionalities such as fusing, filtering and splitting that also occur in database systems, rule-based systems and business intelligence systems; therefore some notations for specifying event processors use extensions of SQL, or rules, or statistical packages. Specifications of responder subsystems identify activities carried out when an event is detected; these activities are often specified using notations for business process management (BPM) and business application monitoring (BAM).

Key Applications

Many areas of science, commerce and education use event-driven applications. All systems in which data is produced continuously and which require timely response to critical events are candidates for event-driven architectures. Rapid accurate response is a competitive advantage in financial trading of stocks, commodities and foreign exchange. EDA is also used in responding to fraud and non-compliance to regulations.

Cyber infrastructure – the integration of physical infrastructure with information technology – is based on event driven applications. Supply chain applications including airline, railroad and trucking systems use EDA. Energy management in the electrical power transmission system is an increasingly important application of EDA as smart meters and sensors are deployed in homes. Many defense department and homeland security applications, such as interdiction of radiation material, use EDA. Crisis management systems, in public and private sectors, are based on EDA.

Customer relationship management (CRM) requires rapid detection and response to customer interests while customers visit Web sites or call service centers. Event driven applications are used in healthcare to monitor patients and to help an aging population live independently for longer.

Future Directions

Event driven applications will become widespread as sensors become ubiquitous and rates of data production in blogs, news, videos, and other online sources continue to increase. A subdiscipline of event driven systems will be formed by integrating concepts from disciplines such as databases, business intelligence and statistics, distributed systems and control, signal processing and state estimation, decision support and operations research, and collaboration technologies.

Cross-references

- ▶ Active Databases
- ▶ Ad-hoc Queries in Sensor Networks
- ▶ Business Intelligence
- ▶ Complex Event
- ▶ Continuous Query
- ▶ Data Mining
- ▶ ECA Rules
- ▶ Event-Driven Business Process Management
- ▶ Message Queuing Systems
- ▶ Publish/Subscribe
- ▶ Streaming Database Systems
- ▶ Text Mining

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data streams. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 2002.
2. Chandy K.M., Charpentier M., and Capponi A. Towards a theory of events. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, pp. 180–187.

3. Deshpande A., Guestrin C., Madden S.R., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
4. Etzion O. Semantic approach to event processing. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, p. 139.
5. Luckham D. The Power of Events: The Power of Complex Event Processing Addison Wesley, Reading, MA, USA, 2002.
6. Muhl G., Fiege L., and Pietzuch P. Distributed Event Based Systems. Springer, Berlin, 2006.
7. Schulte R. The Business Impact of Event Processing: Why Mainstream Companies will soon use a lot more EDA. In Proc. IEEE Services Computing Workshop, 2006, pp. 51.
8. Zhao F. and Guibas L. Wireless sensor networks: an information processing approach. Morgan Kaufmann, Los Altos, CA, 2004.

Event Driven Service-oriented Architecture

► Event Driven Architecture

Event Emitter

► Event Source

Event Extraction

► Video Scene and Event Detection

Event Flow

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event causality graph

Definition

An event flow is a partially ordered graph that traces event instances and causality relations among them.

Key Points

When an event instance is produced by an event source [1] and transferred to the event processing network, an entry node is created into the "event flow" of the specific application, every event created by an event processing agent is added to the event flow, such that, if the input event instances to the event processing agent are <ei_1,...,ei_n> and the output event instances are <eo_1,...,eo_m> then an edge in the event flow is created for all pairs of the type.

<ei_i, eo_k> where i = 1,...,n and k = 1,...,m. This denotes that there is some causality between the input event-instance and the output event-instance. An event flow may go beyond a single event processing network [3]; an input event to an orchestration node in a sink triggers an action, which in turn, creates more events.

Event flows [2] are being used for purposes of validation for event processing applications, as well as for event lineage.

Cross-references

► Event Causality
► Event Lineage
► Event Processing Agent

Recommended Reading

1. Luckham D, Schulte R. (eds.). EPTS Event Processing Glossary version 1.1 http://complexevents.com/?p=409
2. Schwanke R.W., Toward a Real-time Event Flow Architecture Style. In Proc. 8th Annual Int. Conf. and Workshop on the Engineering of Computer-based Systems, 2001, pp. 94–102.
3. Sharon G., Etzion O. Event Processing network: Model and implementation. IBM Syst. J., 47(2):321–334, 2008.

Event in Active Databases

AnnMarie Ericsson, Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

Events in active databases are typically considered to be atomic and instantaneous, i.e., they either happen or not. Events are either primitive or composite, that is, either they are part of the system (primitive) or they are specified in terms of other events (composite).

Key Points

An event is a happening of interest that can invoke execution of reactive behavior, for example, trigger an ECA rule. Events in active databases are typically considered to be atomic and instantaneous [1,3,4], that is, they either happen or not. However, this leads to problems [2] addressed in the event specification entry.

Events can be classified as primitive or composite events. Primitive events refer to elementary occurrences that are predefined in the system. Primitive events are typically further categorized as database events, time events, transaction events, method events etc. A composite event is a set of events matching an event specification.

Events can have attributes, that is, event parameters. Event parameters are usually passed on to the condition and action part of an ECA rule. In general, event parameters can be broadly classified into: (i) System level information, and (ii) Application level information. System level information is related to parameters such as transaction identity, user identity and time stamp(s). Application level information in an object-oriented environment can, for example, include object identity and parameters of the invoked method.

Events play a key role in active databases, since they specify when an ECA rule is triggered. Whenever an event is defined and used in an ECA rule, one can say that the event is explicitly defined. If the active database allows the event to be implied by the rule definition, then the event is said to be implicit. In this case, the event specification of a rule is optional.

Cross-references

- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Atomic Event
- ▶ Composite Event
- ▶ ECA Rules
- ▶ Event specification

Recommended Reading

1. Chakravarthy S. and Mishra D. Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 14(1):1–26, 1994.
2. Galton A. and Augusto J. Two approaches to event definition. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 547–556.
3. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994, Verlag Dr. Kovac, Hamburg, Germany.
4. Gehani N., Jagadish H.V., and Smueli O. Event Specification in an active object-oriented database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 81–90.

Event in Temporal Databases

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Event relation; Instant relation

Definition

An *event* is an instantaneous fact, i.e., something occurring at an instant of time.

Key Points

Some temporal data models support valid-time relations where tuples represent events and where the tuples are thus timestamped with time values that represent time instants. When granules are used as timestamps, an event timestamped with granule t occurs, or is valid, at some instant during t.

It may be observed that more general kinds of events have also been considered in the literature, including events with duration and complex and composite events.

Cross-references

- ▶ Temporal Database
- ▶ Temporal Granularity
- ▶ Time Instant
- ▶ Valid Time

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 406–413.
2. Events. The Internet encyclopedia of philosophy. Available at: http://www.iep.utm.edu/e/events.htm.
3. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version.

In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

Event Lineage

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms
Event pedigree

Definition
An Event lineage is a path in the event processing network that enables the tracing of the consequences of an event, or finding the reasons for a certain action.

Key Points
An event lineage is an audit feature that is determined to answer one of two questions:

1. What were all the implications of event e?
2. What were the events and processing agents that preceded action a?

The event lineage is obtained by maintaining the history of event processing networks using various techniques.

Cross-references
- ▶ Data Lineage
- ▶ Event Flow
- ▶ Event Processing Agent
- ▶ Event Processing Network

Recommended Reading
1. Benjelloun O., Sarma A.D., Halevy A.Y., Theobald M., and Widom J. Databases with uncertainty and lineage. VLDB J., 17(2):243–264, 2008.
2. Sharon G. and Etzion O. Event processing networks – model and implementation. IBM Syst. J., 47(2):321–334, 2008.

Event Mapping

- ▶ Event Transformation

Event Network Edge

- ▶ Event Stream

Event Pathway

- ▶ Event Channel

Event Pattern Detection

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms
Event composition (partial overlap)

Definition
Event Pattern detection is a computational process in which a collection of events are evaluated to check whether they satisfy a pre-defined pattern. Formally an event pattern (EP) is defined as:

EP = <C, IE, PT, PRED, Policies, DER>

where:

- C = Context
- IE = List of Input Event types
- PT = Pattern Type
- Pred = Predicate
- Policies = Semantic fine-tuning policies
- Der = Derived Events

– A context is a collection of semantic dimensions within which the event occurs. These dimensions may include: temporal context, spatial context, state-related context and reference-related context.
– List of input event types provide the EPN edges (event pipes/event streams) that are potentially input to the pattern. Note that events that open or close contexts are indirectly also input event to the patterns within that context
– Pattern Types:

A pattern type is a formula with variables substituted by event types, example- and (e1, e2) – where e1, e2 are

variables of type event type. There are serial pattern types:

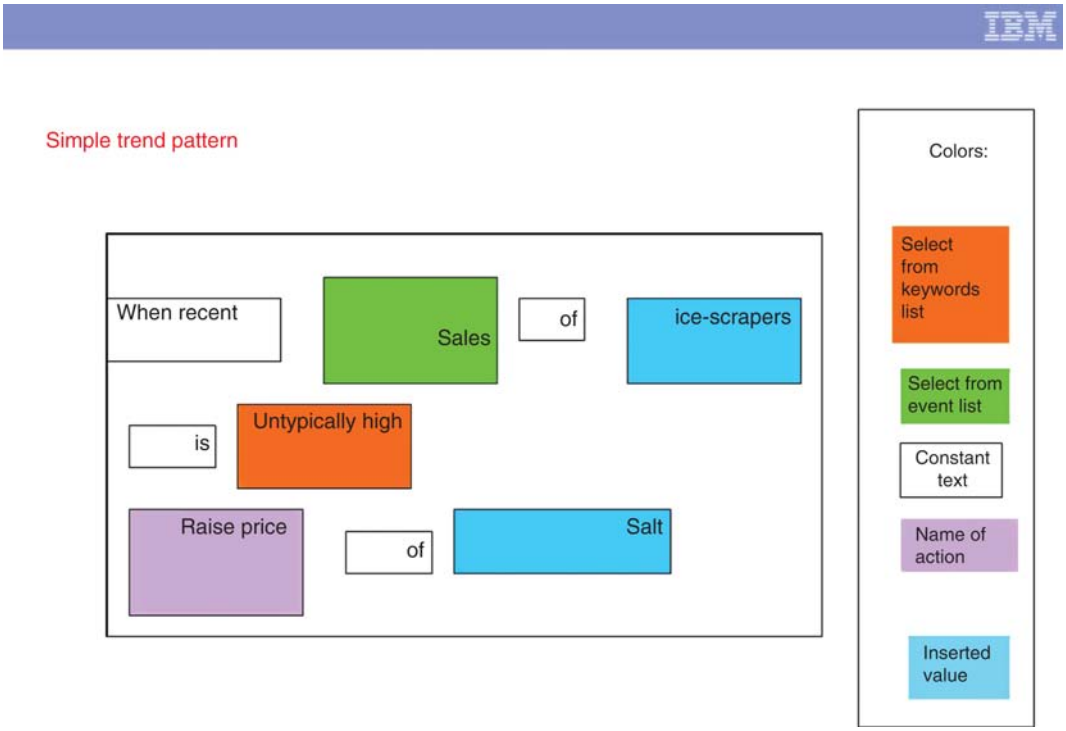
- Basic event oriented patterns, example: and (e1, e2).
- Basic set (stream) oriented patterns, example: (average reading.temperature >35).
- Dimensional patterns – temporal, spatial, spatio-temporal example: moving (e, north).
- Context termination patterns, example: absent (e1).
- Interval-oriented patterns, example: overlap context.

– Predicates: Predicates among various of the input events, example e1.a > e2. b.
– Policies: The policies have been determined to fine-tune the semantics. Example: what should be done if there is synonym events, e.g., the pattern type is and (e1, e2) and there are three instances of e1 followed by five instances of e2 – how many times this pattern will be detected and with which event instances.
– Derived Events: derive one or more events out of the input events that have participated in the pattern detection. The derived event can be a collection of events, or a single event that is derived from the content of other events.

Historical Background

Pattern detection has its roots in several different areas:

1. *Discrete event simulation*: Automatic analysis of event trends to find causality relations among the events in simulation trace has resulted in the need to find "event causality" relations, which has been the motivation for introducing patterns. David Luckham's "complex event processing" has grown up from research in that area
2. *Network and system management*: Network (and later system) management attempt to find the "root cause" of a problem, by taking as an input a collection of symptoms (e.g., device time-outs) and diagnose the core problem that has caused these symptoms. There are relatively simple patterns that belong to this family
3. *Active Databases*: The work on "composite events" in active database is one of the example of event patterns, it features some forms of patterns, and also some policies ("consumption modes")
4. *Middleware*: Publish/subscribe with content selection, and ESB mediations are sources of further (relatively basic) patterns



Event Pattern Detection. Figure 1. User oriented patterns.

Foundations

The notion of pattern has originated from building architecture, as a tool to capture architectural decisions. The area of software design has borrowed this term to document a collection of activities which issue the "best practice" to solve a particular issue (Figure 1). In event processing, the notion of patterns may have several interpretations:

1. Detection of patterns over the event history – as referred to in this entry
2. User-oriented templates that may be developed into more than a single pattern.
3. Best practices pattern of design of use.

Key Applications

Patterns can be related to various types of applications:

1. Real-Time Enterprise application: in which patterns are identified to create actions that are flowing back into the business logic and effecting it. Example: algorithmic trading, active liquidity management, various applications of autonomic computing.
2. Business Activity Management: in which patters are identified to find threats, opportunities and exceptional deviations from the norm or desired behavior measured by constraints and Key Performance Indicators.
3. Diagnostics application: In this application events are symptoms and pattern detection is used in order to determine the problem root cause.
4. Information dissemination: In some cases patterns are used to enable more sophisticated subscription systems, where the user wishes to subscribe not to raw events but to cases in which patterns are satisfied, example: IBM stock has gone up in 2% within 2 h.
5. Prediction: In this case there should be a support in predicted events and the pattern can include predicted events, where the reaction intends to mitigate or eliminate some situation from occurring.

Cross-references

- ► Complex Event
- ► Complex Event Processing
- ► Composite Event
- ► Context

Recommended Reading

1. Buschmann F., Henney K., and Schmidt D.C. Pattern-Oriented Software Architecture, a Pattern Language for Distributed Computing, vol. 4. Wiley, New York, 2007.
2. Coplien J.O. and Schmidt D.C. (eds.). Pattern Languages of Program Design. ACM Press/Addison-Wesley, Reading, MA, 1995.
3. Etzion O. Event processing, architecture and patterns, tutorial. In 2nd Int. Conf. on Distributed Event-based Systems, 2008.
4. Gawlick D. and Mishra S. CEP: functionality, technology and context, Tutorial. In 2nd Int. Conf. on Distributed Event-based systems, 2008.
5. Hoope G. and Woolf B. Enterprise Integration Patterns – Designing, Building, and Deploying of messaging solutions. Addison-Wesley, Reading, MA, 2003.
6. Luckham D. Power of Events, Addison-Wesley, Reading, MA, 2002.
7. Römer K. Discovery of frequent distributed event patterns in sensor networks. In Proc. 5th European Conf. on Event Patterns in Wireless Sensor Networks, 2008, pp. 106–124.
8. Sharon G. and Etzion O. Event processing networks – model and implementation. IBM Syst. J., 47(2):321–334, 2008.
9. Widder A., von Ammon R., Schaeffer P., and Wolff C. Identification of suspicious, unknown event patterns in an event cloud. In Proc. Inaugural Int. Conf. on Distributed Event-Based Systems, 2007, pp. 164–170.

Event Pedigree

► Event Lineage

Event Pipe

► Event Stream

Event Prediction

Segev Wasserkrug
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Prediction regarding future events; Prediction of event occurrence

Definition

Event prediction is the capability to estimate the possibility that some event may occur in the future. Some applications may even be required to advance beyond

simply predicting the occurrence of events and need to influence the occurrence of some future event. For example, this may involve either decreasing the chance that a non-desirable event will occur or increasing the chance that a desirable event will occur. To determine whether or not such actions should indeed be carried out requires decision-making capabilities.

Historical Background

The first event-based systems were active databases in which automatic actions were carried out as a result of database queries. This was done using the *ECA* (Event-Condition-Action) paradigm. As such applications evolved, the events to which applications were required to respond evolved beyond single queries (e.g., insertion of deletions of data) and needed to be deterministically inferred from several such queries. As the complexity of such applications continued to grow, and as event-based applications expanded into other application domains, the need to handle explicit uncertainty regarding the occurrence of events needed to be accommodated. Event-based applications needed to handle unreliable sensor readings and reason about the occurrence of events that could not be deterministically inferred.

The next evolutionary step to such event-based capabilities is to enable event-based systems to reason about the possibility that events will occur in the future and provide facilities for reasoning about possibly influencing such occurrences.

Foundations

Events that may or may not occur in the future are inherently uncertain. Therefore, in many cases, event-based applications are required to estimate the likelihood that such events will occur and attempt to influence this likelihood.

As an example, consider an e-trading web site from a commercial bank. Customers interact with this kind of site using banking transaction events, which include queries regarding the account balance, withdrawals, and deposits. Such events may serve as indications of future events. For example, there may be a number of withdrawal actions carried out by a customer, the end result of which is that there is little or no money left in the account. This set of events may be an indication of the customer's intent to close the account.

In such a case, it would be beneficial for the bank to be able to predict the likelihood that the customer will indeed close the account based on the set of withdrawal events. Furthermore, such an application should be able to consider how possible actions could potentially reduce the likelihood of such an event, while taking into account the potential costs and benefits involved.

As the above example illustrates, there are two main components to a solution that addresses event prediction:

- Component enabling prediction regarding future events
- Component that facilitates decisions regarding actions intended to impact the likelihood of future events, while accounting for the implications of these actions

Mechanisms for Event Prediction

A variety of data can be associated with the occurrence of an event (this data is called the event payload). Two examples of such data are the point in time at which an event occurred and the new stock price in an event describing a change in the price of a specific stock. Some data are common to all events (e.g., their time of occurrence), while others are specific only to some events (e.g., data describing stock prices are relevant only for stock-related events). The data items associated with an event are termed *attributes*. Because an event has attributes, the prediction of an event needs to determine the likelihood of the event occurring as well as the possible values of its attributes.

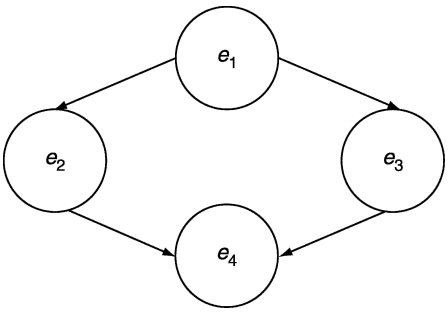
The definition of an event is very broad and may encompass a wide variety of information. Therefore, there are many relevant mechanisms from other domains that can be applied to event prediction. However, when applying these mechanisms, characteristics specific to event-based applications must be taken into account. These characteristics include the need to predict the attribute values of future events and the ability to take into account causal relations between events. This is elaborated below.

Because each event has a set of attributes associated with it, future predictions regarding an event must be able to provide some information about the possible values of the attributes, as well as some information regarding the likelihood that a specific attribute will indeed be assigned a specific value.

The need to take into account causal relations between events stems from the fact that the occurrence of some events may provide information regarding the occurrence of other events. For example, it may be the case that if an event e_1 occurs, event e_2 becomes more likely to occur as well. Such causal relations could be much more complex. For example, consider the relation depicted in Fig. 1, which states the following:

- Event e_1 occurring increases the likelihood of event e_2 occurring, as well as the likelihood of event e_3 occurring.
- Event e_2 occurring increases the likelihood of event e_4 occurring.
- Event e_3 occurring also increases the likelihood of event e_4 occurring.

These relations between events need to be taken into account in the correct manner. For example, if the likelihood of events e_2 and e_3 occurring increase independently of each other, the likelihood of event e_4 occurring may be greater than the case in which the likelihood of events e_2 and e_3 occur increases due only to the fact that event e_1 has occurred.



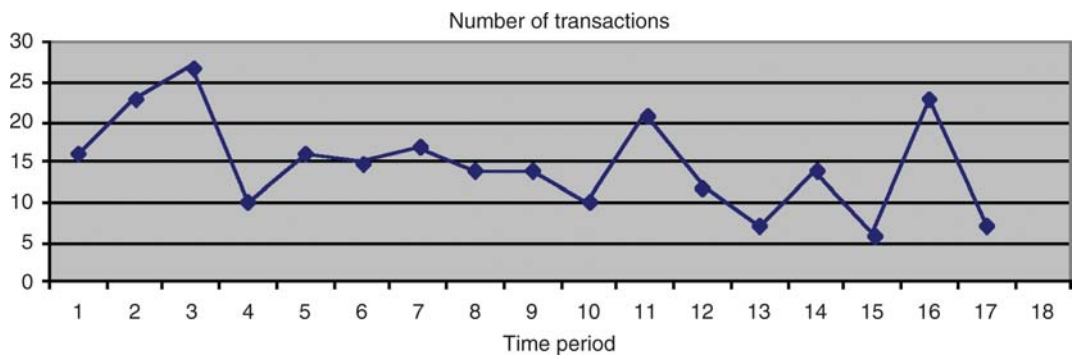
Event Prediction. Figure 1. Example of causal relations between events.

Methods for event prediction can be divided into two general types: explicit modeling methods and implicit modeling methods.

Explicit modeling methods are methods that require the ability to explicitly state the relationship between existing knowledge and events which must be predicted. Usually, such relationships take some known functional form. An example of a well known explicit modeling method is time series analysis, in which some future value is estimated, based on past values, using some function of the past values. The form of the function is defined by the person building the time series model, while the parameters of the function are derived from past data.

Figure 2 shows a plotting of past time series data in which the number of transactions is plotted for 17 time points. Based on this data, the number of transactions at the 18th time point can be predicted using time series analysis methods.

Another explicit modeling method is event composition languages. Event composition languages enable the definition of rules regarding the relationship between events. While most event composition languages are deterministic, there are event composition languages that can take uncertainty into account. Some of these languages are also suitable for defining the relationships required to enable event prediction. For example, a rule could state that if event e_1 is known to have occurred, the probability of event e_2 occurring at some future time t is 0.7. Such languages have the added benefit of naturally enabling the representation of the causal relationships between events. Furthermore, if the language is general enough, it can represent any functional form. Therefore, such languages can be seen as a generalization of all explicit modeling methods.



Event Prediction. Figure 2. Time series plot.

Implicit modeling involves models in which the relationship between the evidence and future events are not explicitly stated, but are directly derived from historical data. An example of such a modeling method is neural networks from the machine learning domain. In the neural network model, a model is constructed which may take many functional forms. The specific functional form is then derived from historical data.

The advantage of explicit models is that they can capture any inherent relationships between existing knowledge and future events, and utilize this knowledge to enable more accurate predictions. The disadvantage is that the person creating the models must possess this knowledge and be able to articulate it in a formal manner. Implicit models remove this burden from the model creator, but require large amounts of historical data. In addition, in implicit models it is more difficult to utilize inherent knowledge to guide the predictions. Consequently, there are also models that attempt to combine the two approaches by enabling the utilization of inherent knowledge when such knowledge is readily available, while deriving other parts of the model from historical data.

Influencing the Occurrence of Future Events

A solution for influencing the occurrence of future events must take into account the following:

- A set of alternative actions A, where each action $a \in A$ is intended to influence the event.
- For each action $a \in A$, the cost c_a of taking the action.
- For each action $a \in A$, the manner in which a will influence the probability of the event.
- For an undesirable event, the benefit b if the event does not occur.
- For a desirable event, the benefit b if the event occurs.
- For an undesirable event, the penalty p if the event occurs.
- For a desirable event, the penalty p if the event does not occur.

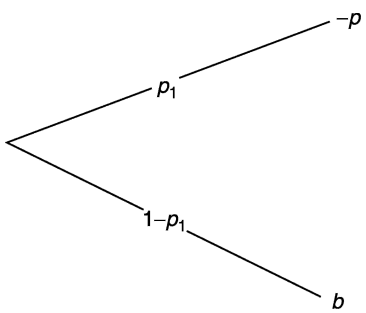
For example, consider the event of a customer closing a banking account. One possible way to prevent a customer from closing the account is to offer better terms, such as higher interest rates on savings. Offering such terms is an action $a \in A$. The loss of income due the higher interest rates is the cost c of taking action a. The benefit of the customer not closing the account is b, while the penalty p is the penalty associated with the customer closing the account.

Several methods exist for taking into account criteria such as the one above. Perhaps the most well known method is utility theory, where decisions are made based on the maximization of some utility function u. The possible decisions in utility theory are often represented by decision trees, such as the one in Fig. 3, which represents the decision tree associated with the event of a customer closing an account.

In Fig. 3, the two branches represent the two possibilities: The customer closing the account (the top branch), and the customer not closing the account (the bottom branch). The quantities at the end of the branches represent the penalties and benefits associated with each action. Therefore, the quantity at the end of the top branch is $-p$, which is the penalty when the customer closes the account, while the benefit of the bottom branch is b, i.e., the benefit when the account remains open. Note that this decision tree represents the case in which no action is taken. In this case, the expected utility can be calculated by $p_1 \cdot (-p) + (1 - p_1) \cdot b$.

Consider the case where it is expected that action a will reduce the probability of the customer closing the account from p_1 to p_2, such that $p_2 < p_1$. For this second case, the expected utility is $p_2 \cdot (-p) + (1 - p_2) \cdot (-c + b)$. In such a case, the decision may be to carry out action a whenever $p_2 \cdot (-p) + (1 - p_2) \cdot (-c + b) > p_1 \cdot (-p) + (1 - p_1) \cdot b$.

Note that the above mechanism for decision making does not explicitly represent an action $a \in A$. Rather, this representation was implicit in the different probability spaces and benefits/costs. Other decision mechanisms, such as Markov Decision Processes (MDPs), do enable an explicit representation of the actions involved, the



Event Prediction. Figure 3. Sample utility decision tree.

manner in which the actions influence the probability of an even occurring, and the associated costs and benefits.

Key Applications

Event-based applications that only react to past events are termed *reactive applications*. Applications that predict, and act upon the prediction of future events, are said to possess *proactive functionality*. Proactive functionality may be required in almost all domains. One such domain is Customer Relationship Management (CRM), in which proactive actions are often required to retain and/or grow market share and benefit. The banking example used throughout this entry is an example of an application in the CRM domain.

Future Directions

While research in event-based applications has been ongoing for several years, research in event prediction is still in its infancy. Therefore, this field is still largely unexplored and is ripe for future research.

Cross-references

- ▶ Active and Real-time Data Warehousing
- ▶ Active Database (aDB)
- ▶ Active Database (management) System (aDBS/aDBMS)
- ▶ Atomic Event
- ▶ Complex Event Processing
- ▶ Complex Event
- ▶ Composite Event
- ▶ Event
- ▶ Event and Pattern Detection over Streams
- ▶ Event Causality
- ▶ Event Driven Architecture
- ▶ Event Pattern Detection
- ▶ Explicit Event
- ▶ Implicit Event
- ▶ Uncertainty in Events

Recommended Reading

1. Alpaydin E. Introduction to Machine Learning. MIT Press, Cambridge, MA, 2004.
2. Brockwell P.J. and Davis R.A. Introduction to Time Series and Forecasting. Springer, New York, 2002.
3. Fishburn P.C. The Foundations of Expected Utility. Kluwer, Dordrecht, 1982.
4. Halpern J.Y. Reasoning about Uncertainty. MIT Press, Cambridge, MA, 2003.
5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Wasserkrug S., Gal A., and Etzion O. A model for reasoning with uncertain rules in event composition systems. In Proc. 21st Annual Conf. on Uncertainty in Artificial Intelligence, 2005, pp. 599–608.

Event Processing

- ▶ Complex Event Processing

Event Processing Agent

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event processing component; Event processing mediator

Definition

A software module and a node in the event processing network that process events [2].

Key Points

An event processing agent [3] is a node in the event processing network that receives one or more events as input, processes them, and creates one or more events as output. There are various types of event processing agents [4]:

1. Filter agent: receives a single event and decides based on a predicate expression whether to route this event further or not (in this case it is a terminal node for this specific event instance).
2. Validation agent: receives a single event and a predicate expression that serves as an integrity constraint for this event. If the integrity constraint is violated may act as a filter (reject the event), may transform the event, or may orchestrate additional actions in an event sink.
3. Transformation agent: receives a single event and transforms this event to another event format.

4. Split agent: receives a single event and creates multiple events, all are functions of the original event (may be either clones or the same events, or distinct overlapping or non-overlapping events).
5. Aggregation agent: receives multiple events and creates a derived event, whose values are aggregated from the values of the input events.
6. Pattern-detection agent: receives multiple events and based on a pre-defined pattern creates a complex event from a collection of events that satisfy the pattern.

Aggregation and pattern detection are statefull agents, and the rest are stateless agents. Aggregation agent has a small state that is incrementally updated, and the input events are not kept; Pattern-detection agent keeps some or the entire input event during the processing period, thus, it may develop large states. Agents may receive events by push or pull [3].

Cross-references

- ▶ Complex Event
- ▶ Event Pattern Detection
- ▶ Event Processing Network
- ▶ Event Routing
- ▶ Event Transformation

Recommended Reading

1. Luckham D. The Power of Events. Addison-Wesley, 2002.
2. Luckham D., Schulte R. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
3. Loke S.W., Padovitz A., and Zaslavsky A. Context-Based Addressing: The Concept and an Implementation for Large-Scale Mobile Agent Systems Using Publish-Subscribe Event Notification. In 4th Int. Conf. on Distributed Applications and Interoperable Systems, 2003, pp. 274–284.
4. Sharon G., Etzion O. Event Processing network: Model and implementation, IBM Syst. J., 47(2):321–334, 2008.

Event Processing Component

- ▶ Event Processing Agent

Event Processing Mediator

- ▶ Event Processing Agent

Event Processing Network

Guy Sharon
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

EPN

Definition

An event processing network (EPN) is a graph $G = (V,E)$ where:

1. The set of nodes in an EPN graph G is denoted V such that $V = C \cup P \cup A \cup EC$. V is a set of nodes of four types; C denoting Event Consumer, P denoting Event Producer, A denoting Event Processing Agent and EC denoting Event Channel.
2. The set of edges in an EPN graph G is denoted E such that $E = \{(u,v) \mid (u \in (P \vee A) \rightarrow v \in EC) \wedge (u \in EC \rightarrow v \in (C \vee A))\}$. E is a set of ordered pairs of nodes representing directed edges. These edges are between either an event producer to an event channel, an event channel to an event consumer, an event processing agent to an event channel, or an event channel to an event processing agent.

The EPN model also includes a relationship definition between edges called Event Causality.

An event processing network (EPN) describes how events received from producers are directed to the consumers through agents that process these events by performing transformation, validation or enrichment. Any event flowing from one component to another must flow through a connection component. Therefore, an EPN model consists of four components: Event Producer, Event Consumer, Event Processing Agent, and a connection component called Event Channel.

Historical Background

Event driven computation is not a new concept; it is being used in user interfaces to perform a specific code when a button is pushed and when a mouse button is clicked or released. It is also being used in database systems as triggers to perform specific logic when a record is inserted, deleted or updated. Some existing approaches in configuring and expressing the event processing directives in event-driven systems are through SQL-like languages, script languages, or rule languages and are executed by standalone software, messaging

systems or data stream management systems. While event driven processing is not a new concept, there is a need to provide conceptual techniques for definition and modeling of an event driven architecture, making it possible to express event processing intentions independently to the implementation models and executions. This is done through a conceptual model called Event Processing Network [3].

Foundations

As businesses are required to be more agile, to respond to the market rapidly and to be able to adapt quickly and efficiently, there is need for underlying systems and business applications to support such an environment. A sound and cost-effective way to achieve such goals is to introduce an event-driven architecture (EDA).

The producer (source) of the events only knows that the events transpired. The producer has no knowledge of the events' subsequent processing, or the interested parties, leading to entirely decoupled architecture. Such architecture enables flexible definition of the event processing logic: detection of event patterns, derivation of new events, transformation, and routing from producers to consumers based on the business logic required. Thus businesses can react to changes, execute relevant processes as a result, and influence ongoing processes based on the changes. Furthermore, such definition of event processing can be easily modified and quickly deployed in accordance with business needs such as changes to business processes and policies. The Event Processing Network (EPN) concepts model these processing directives.

An EPN model consists of four components: Event Producer, Event Consumer, Event Processing Agent, and a connection component called Event Channel.

Event Channel

An event channel [4] is a mechanism for delivering event streams from event producers and event processing agents to event consumers and event processing agents.

The event channel may receive multiple event streams from different sources and may transfer a combined event stream from all the sources to multiple sinks. It is the responsibility of the event channel to create an ordered and combined event stream from the sources and provide this stream to each target.

Another responsibility of an event channel is to retain the history of the events streaming through for retrospective event processing [6], defined as the discovery of event patterns over the history of events as opposed to online event processing, detecting predefined event patterns as new events become available.

Within a network of nodes and edges an event channel is represented as a node with edges directed to and from the node. Every incoming edge represents an event stream from an event producer or processing agent to publish events on to the channel. Every outgoing edge represents an event stream to an event consumer or processing agent subscribed to the channel. In Fig. 1 a graphical representation of an event channel is presented, where the name of the channel is within the boxed arrow and the event types are given on top of each edge to signify what type of events are provided by the event stream.

An Event Channel is defined as follows:

1. Every node in *EC* may have event retention policies defined for retrospective processing.
2. The edges connected to a node in *EC* are defined as $\forall v \in EC,\ \exists e_{in}, e_{out} \in E\colon\ e_{in} = (u \in (P \vee A),\ v),\ e_{out} = (v,\ u \in (C \vee A))$.

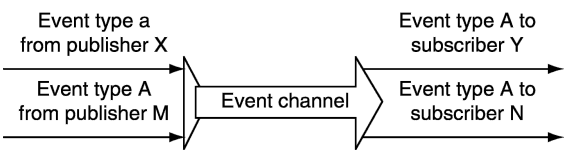
Event Producer and Consumer

An event producer, also known as an event source or event emitter, produces and publishes events through event channels for any party of interest to consume. This could be an event consumer or an event processing agent.

Within a network of nodes and edges an event producer is represented as a source node, i.e., there exist only edges directed from it. The number of edges directed from it is the number of different event channels that the producer publishes to. The edges connected to a node in P are defined as $\forall v \in P,\ \exists e_{out} \in E\colon\ e_{out} = (v,\ u \in EC)$ (Fig. 2).

An event consumer, also known as an event sink, is interested in events to perform its responsibilities. Once the event of interest is known to the consumer it will perform a certain task associated with this event.

Within a network of nodes and edges an event consumer is represented as a sink point, i.e., only edges directed to it. The number of edges directed to



Event Processing Network. Figure 1. Event channel.

it is the number of different event channels that the consumer subscribes to. The edges connected to a node in C are defined as $\forall v \in C, \exists e_{in} \in E$: $e_{in} = (u \in EC, v)$ (Fig. 3).

Event Processing Agent

There are cases where there is a gap between the events produced by event producers, and the events required by the event consumers. These events may have different syntax (structure) and/or different semantic meaning than expected. There are also cases where a single event will not trigger an action performed by an event consumer, instead, by a complex composition of events happening at different times and in different contexts [2]. Event processing agents (EPA), also know as event mediators, are needed to detect patterns on 'raw' events, then to process these events through enrichment, transformation and validation, and finally to derive new events and publish them. An event processing agent is responsible in producing these derived events and decides where and how these events should be published.

The event processing agent is made up of three stages:

- Pattern detection – this stage is responsible in selecting events for processing according to a specified pattern.
- Processing – this stage is responsible in applying the processing functions on the selected events satisfying the pattern, resulting in derived events.
- Emission – this stage is responsible in deciding where and how to publish the derived events.

An event processing agent subscribes to event channels for its pattern detection and publishes to event channels while emitting events.

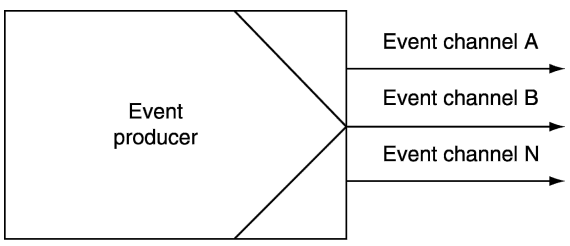
Within a network of nodes and edges an event processing agent is represented as a node with edges directed to and from the node. The number of edges directed to it is the number of different event channels that the agent subscribes to for detecting the pattern. The number of edges directed from it is the number of different event channels that the agent publishes to based on the processing and emission definitions.

The edges connected to a node in A are defined $\forall v \in A$, $\exists e_{in}, e_{out} \in E$: $e_{in} = (u, v)$, $e_{out} = (v, w)$, $u, w \in EC$. e_{in} is an event stream from an event channel the EPA is subscribed to and e_{out} is an event stream of derived events to an event channel the EPA publishes to (Fig. 4).

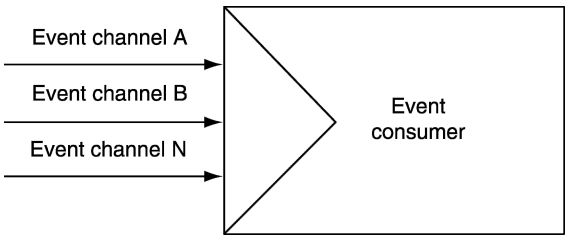
The following sub-sections discuss in detail the different stages within an event processing agent.

Pattern Detection At the pattern stage of an agent, a pattern is defined to be detected at runtime. A single event may match the pattern. In cases where more than one event is needed to detect a pattern, the collection of these events is called a complex event. The single or complex event is passed to the processing stage of an agent. The pattern description consists of four aspects and all are taken from the discipline of complex event processing [1,5] (Fig. 5).

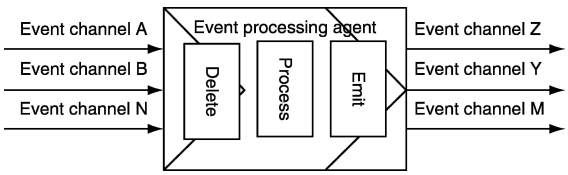
Context – specifies the relevance of the events participating in pattern detection whether it is temporal based, spatial based, or semantic based. Temporal based means that events have to occur within a definable time frame. Spatial based means within a definable space concept such as geographical locations or regions. Semantic based context means that the events participating



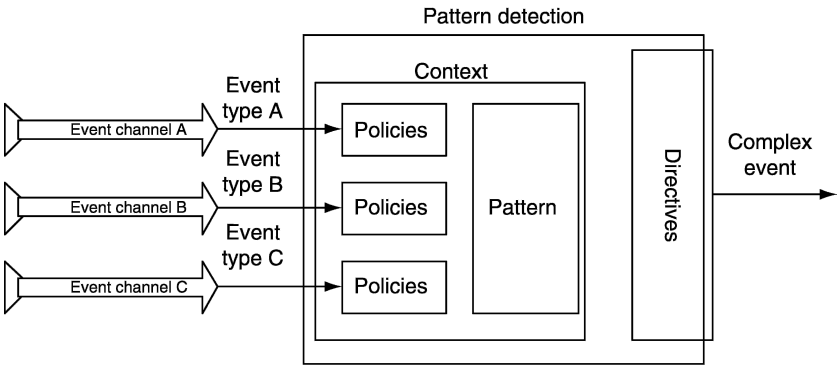
Event Processing Network. Figure 2. Event producer.



Event Processing Network. Figure 3. Event consumer.



Event Processing Network. Figure 4. Event processing agent.



Event Processing Network. Figure 5. Aspects of the pattern detection stage.

in pattern detection have relevance through a mutual object or entity. All types of contexts can be used in the same pattern detection and possibly more than once. Thus, an intersection or union of time frames can be specified as a context, or a context may be defined where events must occur in the same region and within the same time frame for pattern detection. Events occurring within the context defined are denoted as candidates to pattern detection.

Policies – specify how cases in which the semantics need to be fine tuned are handled, discussed in [5] as situation quantifiers. Policies include decisions such as:

- Whether to use the first, last or each of the events in the stream to detect the pattern or any other set operations
- Use only events satisfying a predicate on their attributes
- Whether each event in the stream may override the previous event and remove it from contributing to a pattern detection
- Expiry time, relative or absolute, of an event contributing in detecting the pattern
- Remove an event's contribution, due to the occurrence of a converse event, an event which conditionally implies the expiration of another event, described as contradiction elements in [1]

Choosing each event to contribute to pattern detection may result in multiple pattern detection, resulting in multiple complex events and therefore may result in multiple event publication by the event processing agent as a reaction to a single event consumed by the agent. Events occurring within the context defined and adhere to the defined policies are denoted as candidates to pattern detection.

Pattern – specifies the relationship among events complying with policies within context that is to be detected. Patterns are described in a form of event algebra discussed in [1] as situation operators. Some examples to operation semantics are

- *Any* – any event that occurs within the specified context and for which the policy holds, i.e., candidate, results in pattern detection.
- *Collect* – all events that occur within the specified context and for which the policies hold, i.e., candidates, are collected into one complex event at the end of the detection interval, for example, end of time window and exiting a geographical area.
- *Determine if* – similar to collect, however, the complex event will only include a statement of true if there were events within the detection interval and false otherwise.
- *And {eT1,...,eTn}* – one event from each event type, denoted as *eTi* must occur within the context for this pattern to be detected. There may be multiple pattern detections if there is more than one event of each type. This is determined by policies and whether the occurring events are candidates for pattern detection.

The EPN model does not restrict the set of operations to express patterns nor does it restrict the semantics given to operators.

Directives – specify directives for reporting on pattern detection to the processing stage and what to do with the events that contributed to the pattern detection, discussed in [1] as situation triggering expressions.

- Immediately as the pattern is detected.
- At the end of the detection interval.

- At time-outs associated with event attributes.
- At specifiable periods reporting on all the pattern detections since the last period.
- Allow a single or multiple activation of the processing stage when more than one pattern is detected during the detection interval.
- Consume, maintain always, or maintain an event for certain number of times after pattern detection for further detections, i.e., is the same event allowed to contribute in further pattern detection.
- Include the events that have not contributed to the pattern's detection with the report to the processing stage. When no pattern detection occurred during the reporting time such as at end of detection interval or at time-outs, all events within this period may be passed to the processing stage as well.

Processing Once the pattern detection stage has detected a pattern, it creates either a complex event, in case more than one event contributed to the detection, or passed the original event, in case only it was needed for the detection. This event is further processed based on the processing definition configured for the event processing agent. The result of the processing stage is a derived event ready for emission. There are four possible processing methods that can be performed on the result of the pattern detection stage:

- *Transformation* – specifies a transformation function to be performed on complex events to produce derived events. The transformation can cover a many-to-many relationship. Mapping is in the case of one-to-one where each output of the pattern detection is mapped to a single event structure. Splitting is in the case of one-to-many where each output of the pattern detection is split to several event structures, i.e., multiple derived events from a single complex event. Aggregation is in the case of many-to-one where an aggregation function is performed on the collection of events that are included within the complex event and a single event structure is produced with the function's results. Many-to-many transformation is achieved by applying aggregation and then splitting within the same processing.
- *Derivation* – specifies computations to apply on the collection of events reported by a complex event, which covers more than aggregation transformation. The result can be a single or a set of derived events.
- *Enrichment* – specifies directives in enriching the event with data from external sources such as a database. The result is a new event that is derived from the event received at the processing stage. The additional event data may be used to establish the connection to and retrieval of data from the external source.
- *Validation* – specifies constraints that the event must comply with (assertion) and instructions on how to handle the event in case it does not comply and violates the constraints. The decision on how to handle violation may be conditional and may differ by the repair action.
- Alert and let the processing continue.
- Reject and terminate the processing for this event. The event itself or a derived error event may be published to an event channel for error handling.
- Modify the event for the assertion to hold. When this is not possible, treat it in the same manner as rejecting the event.
- Perform change to external data if used in the assertion to make it hold. Again when this is not possible, treat it in the same manner as rejecting the event.

Emission The emission stage is responsible to emit the derived events to appropriate event channels and is configured with decisions to what event channels to emit and when to emit the events, such as immediately when received from the previous stage, periodically, at specific times or with time offsets. Thus, a single derived event may be emitted to more than one event channel.

Key Applications

Information Dissemination\Situation Awareness

Getting the right information in the right granularity to the right person\target at the right time:

Asset Tracking, Advertising context based content, Command and Control.

Active Diagnostics

Diagnose problems based on symptoms and resolve them:

System Management

Real Time Enterprise Reactions to events as part of business transactions – achieving low latency decisions and quick reaction to threat and opportunities:
Claims processing, Utilities grid management

Business Activity Monitoring
Quick observation into exceptional business behavior and notification to the appropriate people\targets:
Continuous Auditing, Fraud Detection

Predictive Processing
Mitigate or eliminate predicted events:
Epidemic control, Stock Trade

Cross-references

- ▶ Complex Event
- ▶ Complex Event Processing
- ▶ Context
- ▶ EDA
- ▶ Event Causality
- ▶ Event Channel
- ▶ Event Driven Architecture
- ▶ Event Processing Agent
- ▶ Event Sink
- ▶ Event Source
- ▶ Retrospective Event Processing

Recommended Reading

1. Adi A. A Language and an Execution Model for the Detection of Reaction Situations, PhD dissertation, Technion – Israel Institute of Technology, 2003.
2. Adi A., Biger A., Botzer D., Etzion O., and Sommer Z. Context Awareness in Amit. In Proc. 5th Annual Int. Workshop on Active Middleware Services, 2003, pp. 160–167.
3. Etzion O. and Sharon G. Event Processing Network – A Conceptual Model. In Proc. 2nd Int. Workshop on Event-driven Architecture, Processing and Systems, 2007.
4. Event Processing Glossary, http://complexevents.com/?p=195#comments
5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Yang Y., Pierce T., and Carbonell J.G. A Study on Retrospective and On-Line Event Detection. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 28–36.

Event Processing Systems

- ▶ Event Driven Architecture

Event Producer

- ▶ Event Source

Event Relation

- ▶ Event in Temporal Databases

Event Service

- ▶ Channel-Based Publish/Subscribe

Event Sink

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event consumer

Definition

An entity that receive events from other entities [1].

Key Points

Event sink are nodes in the edge of the event processing network (receiving only) that receive events from the event processing network, event sinks are typically of two types:

1. Orchestration event Sink: receives an event from the event processing network and determines what action should be triggered (e.g., ECA rule) [2]. An action can be – database update, activating a software module, activating/modifying/terminating a workflow instance [3].
2. Notification event sink: receives the event from an event processing network and sends it to a consumer – a person or a dashboard [4].

Note: the action that is triggered by the event sink may be an event source and emit events to the same event processing network or to another event processing network, this will be reflected in the event causality and event lineage.

Cross-references

- ▶ Event Causality
- ▶ Event Lineage
- ▶ Event Processing Network
- ▶ Event Source

Recommended Reading

1. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
2. Lin Chen, Minglu Li, Jian Cao, Yi Wang. An ECA Rule-based Workflow Design Tool for Shanghai Grid. IEEE SCC 2005, 325–328.
3. Lin D., Sheng H., and Ishida T. Interorganizational Workflow Execution Based on Process Agents and ECA Rules. IEICE Transactions 90-D(9):1335–1342, 2007.
4. Ruby C.L., Green M.L., and Miller R. The Operations Dashboard: A Collaborative Environment for Monitoring Virtual Organization-specific Compute Element Operational Status. Parallel Processing Letters 16(4):485–500, 2006.

Event Source

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event producer; Event emitter

Definition

Event source is an entity that provides event for the processing cycle [3].

Key Points

Event sources are typically initial nodes (sending only) in the event processing networks, and provide events to the network for further processing. Event sources can be of various types:

1. Software module – through instrumentation
2. Sensors [1]
3. Clock/Calendar
4. State observers in workflows (BPM).

An event can be obtained from a source in various modes:

1. Push – the source initiates the event emission.
2. Periodic Pull – a calendar service pulls one or more events from the source (e.g., by query the log) [2].
3. On-Demand Pull – an event processing agent pulls the source using request/reply protocol.

Getting events from event sources to processing typically requires adapter to create the event in the desired format.

Cross-references

- ▶ Event Processing Network
- ▶ Event Sink

Recommended Reading

1. Chakraborty S., Poolsappasit N., and Ray I. Reliable Delivery of Event Data from Sensors to Actuators in Pervasive Computing Environments. In Proc. 21st Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2007, pp. 77–92.
2. Deolasee P., Katkar A., Panchbudhe A., Ramamritham K., and Shenoy P.J. Adaptive Push-Pull: Dissemination of Dynamic Web Data. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 265–274.
3. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.

Event Specification

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

Event declaration; Event definition; Composite event query

Definition

Event specifications define event types in terms of patterns of other event types by using event operators (either expressed as operators in an operator grammar or as functions in a functional grammar) and event contexts. Event types are categorized as atomic (or primitive) or composite. An atomic event type is either a system primitive (for example, begin transaction) or it can be any defined atomic change in an activity such as a method, procedure, task etc. (e.g., method has been called). Event types defined in terms of other event types are named composite event types.

Historical Background

Essentially, whenever something has to be defined, it is necessary to have a specification. This can be found in

several disciplines of computer science, for example, programming languages, compiler technology etc.

Event specifications became significant with the advent of the HiPAC project [10], in which seminal work on composite event types were done. Prior to HiPAC, mainly primitive event types were addressed in research publications and the event specifications only consisted of single atomic event types (often only system primitives).

Atomic event types in active databases can be categorized according to their sources (e.g., cf. Buchmann [4]): transactional events, temporal events, database manipulation events, and user-defined events. Transactional events are usually system primitives denoting state changes such as "transaction has begun," "transaction has committed," and "transaction has aborted." Temporal events are either absolute (a specified point in time) or relative to some other event and denotes that a time point has passed; further, temporal events can be repetitive and a special case of temporal events are periodic events. The database manipulation events depend on whether the database is relational, object-oriented or object-relational. In relational databases, there is typically only system primitives associated with the state changes corresponding to insertion, update, and deletion of entities (in particular, tuples). In object-oriented databases, system primitive events are typically associated with methods in existing classes associated with the database schema. Finally, in object-relation databases, a mix of system primitive events from both the relational aspect as well as the object-oriented aspect can be found. User-defined atomic events are either associated with methods [4] or composite event queries [3].

There are several event specification languages (either as complete specification languages in their own right or as intrinsic parts of another specification language), usually associated with the active database prototype (e.g., Snoop in Sentinel [8,9], SAMOS [12], ODE [13]), but there are several others that are not explicitly associated with active databases (e.g., GEM [16], monitoring of real-time logic [15]). All of these event specification languages in active databases share a basic set of event operators (e.g., Mellin [17], Carlsson and Lisper [6]): sequence, conjunction, disjunction and non-occurrence. The non-occurrence of events is always enclosed in an half-opened time interval, the question is what event types can be employed to delimit the interval. For example, in Snoop [8,9] and its descendants (e.g., REACH [5]) and SAMOS [12] intervals are delimited by arbitrary event types, whereas in ODE [13] the interval is from when the transaction, in which events occurs, begin. The interval is typically closed by an request to commit transaction or a relative temporal event with respect to the beginning of the interval.

Composite events are specified by event expressions in an operator grammar or in a functional grammar. For example, in Snoop [9] a sequence of events of type E_1 followed by events of type E_2 is expressed as $E_1;E_2$, whereas in EPL [18] it is expressed as $SEQ(E_1,E_2)$. Note that this specification does not dictate how many events of each type that matches the specification and it does not dictate if the events of different event types in the specification are interleaved. Most approaches (e.g., HiPAC [10], Snoop [9,8], SAMOS [12], ODE [13]) employ an operator grammar, whereas, for example, EPL (Event Pattern Language) [18] and XChange^{EQ} [3] are based on a functional grammar. Additionally, XChange^{EQ} is based on XML (e.g., Common Base Events [7]). The advantage of a functional grammar over an operator grammar is threefold:

1. There is no problem with associativity (e.g., the expression SEQ (E_1,E_2,E_3) does not have the ambiguity found in $E_1;E_2;E_3$ that either can mean $E_1;(E_2;E_3)$ or $(E_1;E_2);E_3$)
2. There is no problem with operator priority, a problem that is found in any operator grammar. For example, a conjunction should be prioritized over a disjunction so that $E_1 \& E_2 | E_3$ is equivalent to $(E_1 \& E_2) | E_3$ meaning that either an E_3 occurs or a conjunction of E_1 and E_2 occurs, but how should sequences be prioritized with respect to conjunctions and disjunctions?
3. Operators in a functional grammar can be extended to be based on an arbitrary number of event types (e.g., SEQ $(E_1,E_2,...,E_n)$).

There are some pitfalls in event specifications: the definition of disjunctive event operator, and treating all event types as instantaneous. One event operator that have an unintuitive meaning in many event specification languages is disjunction (e.g., Snoop, SAMOS). In logic, disjunction can be inclusive or exclusive, but in many event specification languages it is neither, but a hybrid. Essentially, if an E_1 and an E_2 event occurs concurrently and E **defined by** $E_1|E_2$ is monitored, then the result is two event occurrences of type E with the same end time rather than one

event occurrence of type E (inclusive disjunction) or no event occurrence (exclusive disjunction).

Initially, all events, both atomic and composite, were considered to be instantaneous [8,9]. However, as pointed out by Galton and Augusto [11], if all events are considered to be instantaneous, then this will introduce ambiguities in event specifications. For example, it is impossible to distinguish between E_1; (E_2;E_3) and E_2;(E_1;E_3).

Parameterized events carry parameter apart from the system default parameters of events, where the default parameters typically are event type, time stamps, and scope in which the event was generated (e.g., which transaction was the event generated in). Somehow, the event specification needs to enable specification of event parameters as, for example, by Bækgaard et al. [1]. Further, if parameterized event detection is allowed, then it is necessary to specify what parameters are part of the event type specification. Examples of specification of parameterized event detection are found in TCCS [1], XChange^{EQ} [3], and Solicitor [17].

Foundations

As always in formal specifications, there are two main approaches: either to define a minimal specification language that can handle all relevant situations or to define an expressive specification language in which the available expressions and operators are as close as possible to what is needed to express. The latter is prevalent in active database research (cf. Zimmer and Unland [19]), since it is assumed to lead to more comprehensive event specifications (e.g., the work by Liu et al. [15] illustrates the difference between Snoop and real-time logic)

There are four significant research issues that have been pursued within active databases with respect to event specifications: (i) the performance characteristics of event composition, (ii) how to introduce algebraic properties in event specification languages, (iii) how to define event specification languages that enable comprehensive specifications and, in particular, how to specify the orthogonal concepts of patterns and event contexts; and (iv) tools and techniques for understanding event specifications. Issue (i) has been dealt with within a benchmark named BEAST [14] as well as in algorithmic time complexity studies [17, ch. 9]. In the BEAST benchmark study [14] it was concluded that the underlying mechanism (e.g., finite state automatas, Petri-nets, or arbitrary code) of event composition was insignificant compared to typical information maintenance issues such as indexing of event types. It has been shown that the algorithmic time complexity of event composition is mainly based on event contexts, not the event operators per se [17, ch. 9].

The introduction of algebraic properties in event specification languages has been addressed by Carlsson and Lisper [6]. Essentially, they have introduced algebraic properties to an event specification language by defining a new event context: a variety of the recent event context. It is also an attempt to make a minimal language.

A few studies have addressed the specification events (issue (iii)), in particular how to mix event operators and event contexts. There are three approaches: (a) the event operator is locked to one event context as in, for example, ODE [13] or the work by Carlsson et al. [6]; (b) the event operators are explicitly associated with an event context as in, for example, Solicitor [17]; and (c) the different event types in the expressions are decorated with notation (For example, SEQ($last$: E_1,$first$: E_2) is an example of using the proposed decorators *first* and *last*. This example specification is equivalent to SEQ(E_1,E_2) in recent event context.) that indicate the event context as proposed by Zimmer and Unland [19]. The most comprehensive form is approach (b), but more work is required, in particular on the topic of specifying parameterized event detection. For example, assume that there are concurrent processes accessing a shared variable and it is desirable to check that the processes behave by monitoring that processes enter and exit the critical regions associated with the shared variable within an expected time interval without a "terminate application" event in between. In this case, the matching of events representing entering and exiting the critical regions must be based on that they are generated by the same process, but the "terminate application" event is associated with all existing processes in the application. Further, parameters may express temporal scopes that subsume each other. For example, an event may be generated in a transaction by a particular process executing a specific component and events may carry the identity of transaction, process and component; in some cases, it is only desirable to only match on the transactional level, but in other cases a more precise scope such as the component level is desirable.

Concerning issue (iv), Berndtsson et al. [2] identified, in their investigation of existing visualization techniques of event composition, that event

specifications are difficult to understand for M.Sc. level students. The conclusion is that tools to understand event specifications are necessary. Compared to condition evaluation, event composition is far more complicated since, in addition to condition evaluation, it is necessary to check what happened in previous evaluations of event expressions.

Key Applications

Buchmann [4] claims that there are two reasons for separating events from the condition in a rule: (i) to add the semantics of when the rule should be evaluated from what should be evaluated and (ii) efficiency of rule execution. Essentially, event specifications are employed for rule optimization in active databases. Further, if an active database is part of an event based system, then event specifications can be employed to alert other applications outside the active database.

Future Directions

There are three major unresolved threads of research on this topic: (i) specification of parameterized event types for parameterized event detection; (ii) reuse of event specifications; and (iii) usefulness of algebraic properties. The first thread addresses a problem that is similar to join in SQL queries and unification in Prolog programming language. That is, it deals with specifying that the values of certain parameters of an event should match the values of other parameters of another event in an event specification. For example, $E = E_1;E_2$ **where** $E_1.p_1 = E_2.p_2$ could be a notation stating that event occurrences of E only consists of event occurrences of E_1 and E_2 where the values of parameters p_1 and p_2 match. Making this kind of specification comprehensive, precise, unambiguous, and reusable is an open problem. (Note that the notation here is chosen for the likeness of SQL queries, it is not necessarily the best way to specify parameterized event types.)

Reuse of event specifications has not been addressed within active databases, but is a significant topic in software engineering. It would be interesting to, for example, consider if template event specifications can be useful in active databases. For example, in distributed systems call and reply semantics is employed in, among other things, remote procedure calls, that is, something is called and a reply is expected. Assume that C, R and T are template parameters to be instantiated, then a template for detecting a correct (E_c) or incorrect (E_i) sequence of call and reply within a time window could be as follows (where tps(E) and tpe(E) returns the time point of start and end of E):

$$E_c\langle C, R, T\rangle \textbf{ defined by } (C; R) \textbf{ without } E_i \textbf{ within } [\text{tps(C)}, \text{tps(C)} + \text{T}]$$

$$E_i\langle C, R, T\rangle \textbf{ defined by } \text{non-occurrence of } E_c \textbf{ within } [\text{tps(C)}, \text{tps(C)} + \text{T}]$$

Given this, a notation such as $E_c\langle A, B, 10s\rangle$ can be used to specify that $C = A$, $R = B$ and $T = 10s$. This specification should also introduce $E_i\langle A, B, 10s\rangle$ implicitly.

With respect to the usefulness of algebraic properties, the question is how useful algebraic properties are? As mentioned, Carlsson et al. [6] has shown that it is possible, but how useful is it compared to existing event contexts. This is still an open issue. For example, basing event composition on historical order, as in the chronicle event context, may become complicated, if not impossible.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Atomic Event
- ▶ Composite Event
- ▶ ECA Rules
- ▶ Event
- ▶ Event Detection

Recommended Reading

1. Bækgaard L. and Godskesen J.C. Real-time event control in active databases. J. Syst. Softw., 42(3):263–271, 1997.
2. Berndtsson M., Mellin J., and Högberg U. Visualization of the composite event detection process. In Proc. Int. Workshop on User Interfaces to Data Intensive Systems, 1999, pp. 118–127.
3. Bry F. and Eckert M. Rule-based composite event queries: the language XChangeEQ and its semantics. In Proc. First Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 16–30.
4. Buchmann A.P. Active object systems. In Advances in Object-Oriented Database Systems. A. Dogac, M.T. Ozsu, A. Biliris, and T. Sellis (eds.). Springer-Verlag, Berlin, 1994, pp. 201–224.
5. Buchmann A.P., Zimmermann J., Blakeley J.A., and Wells D.L. Building an integrated active OODBMS: requirements, architecture, and design decisions. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 117–128.
6. Carlson J. and Lisper B. An event detection algebra for reactive systems. In Proc. 4th ACM Int. Conf. on Embedded Software, 2004, pp. 147–154.

7. Common base events. URL http://www.ibm.com/developerworks/library/specification/ws-cbe/.
8. Chakravarthy S., Krishnaprasad V., Anwar E., and S.K. Kim Composite events for active database: semantics, contexts, and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
9. Chakravarthy S. and Mishra D. Snoop: an event specification language for active databases. Knowl. Data Eng., 13(3):1–26, 1994.
10. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC Project: combining active databases and timing constraints. ACM SIGMOD Rec., 17(1):51–70, 1988.
11. Galton A. and Augusto J. Two approaches to event definition. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 547–556.
12. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
13. Gehani N.H., Jagadish H.V., and Schmueli O. COMPOSE – a system for composite event specification and detection. In Advanced Database Concepts and Research Issues. Springer-Verlag, Berlin, 1993.
14. Geppert A., Berndtsson M., Lieuwen D., and Roncancio C. Performance evaluation of object-oriented active database management systems using the BEAST benchmark. Theor. Pract. Object Syst., 4(4):1–16, 1998.
15. Liu G., Mok A.K., and Konana P. A unified approach for specifying timing constraints and composite events in active real-time database systems. In Proc. 4th Real-Time Technology and Applications Symp., 1998, pp. 199–208.
16. Mansouri-Samani M. and Sloman M. GEM: a generalized event monitoring language for distributed systems. IEE/IOP/BCS Distrib. Syst. Eng. J., 4(2):96–108, 1997.
17. Mellin J. Resource-Predictable and Efficient Monitoring of Events. Ph.D. thesis, no. 876, University of Linköping, 2004.
18. Motakis I. and Zaniolo C. Formal semantics for composite temporal events in active database rules. J. Syst. Integrat., 7(3/4):291–325, 1997.
19. Zimmer D. and Unland R. On the semantics of complex events in active database management systems. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 392–399.

Event Stream

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event pipe; Event network edge

Definition

An Event Stream is a collection of ordered events [4].

Key Points

An event stream is a collection of events, with one or more order relations. The naïve order relation is according to the time that the event arrives to the processing system, but this may not reflect the event order in reality [2], which may define an additional order relation. An event stream may be homogenous (all events have the same structure and type) or heterogeneous (include events that have different types). Homogenous streams are also known as *Data-Type Streams*. For practical usages, typically data-type streams are used. An edge in the event processing network, known as *event pipe* typically is a superset of *event stream* [5].

Sometimes event stream is used as a synonym to event pipe. This term has some relation to "data stream," but it is not identical [1], and does not necessarily correspond to SQL implementation. A typical case of event stream is a time-series, and there are some methods to handle out-of-order arrival in such streams [3].

Cross-references

- ▶ Data Stream
- ▶ Event Processing Network
- ▶ Event Flow

Recommended Reading

1. Albers K., Bodmann F., Slomka F. Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems. In Proc. 18th Euromicro Conf. Real-Time Systems, 2006, pp. 97–106.
2. Barga R.S., Goldstein J., Ali M.H., Hong M. Consistent Streaming Through Time: A Vision for Event Stream Processing. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 363–374.
3. Li M., Liu M., Rundensteiner E.A., and Mani M. Event Stream Processing with Out-of-Order Data Arrival. In Proc. 1st Int. Workshop on Distributed Event Processing, Systems and Applications, 2007.
4. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
5. Sharon G., Etzion O. Event Processing network: Model and implementation, IBM System Journal, 47(2):321–334, 2008.

Event Stream Processing

- ▶ Complex Event Processing

Event Stream Processing (ESP)

► Event and Pattern Detection over Streams
► Stream Processing

Event Topic

► Event Channel

Event Trace Analysis

► Event Detection

Event Transformation

Peter Niblett
IBM United Kingdom Limited, Winchester, UK

Synonyms

Event mapping

Definition

In its broadest definition Event Transformation refers to any operation that takes as input a single event message or stream of event messages, and produces a single event message or stream of event messages as its output. As such, this definition encompasses many of the functions provided by an Event Processing Agent in an Event Processing Network, including agents that do complex pattern detection.

There is a somewhat narrower definition in which the operation has to be either a translation, aggregation, split or composition function.

Historical Background

Event translation (transformation of a single input event message into a single output event message) is a special case of message translation – a more general concept where the messages concerned do not necessarily represent events. Message translation has long been used to solve enterprise integration problems where the applications to be integrated have different data models. Enterprise integration sometimes also involves split operations, where input messages are split into multiple outputs. Hohpe and Woolf discuss a number of message translation and splitting patterns in [4].

Event Aggregation and Event Composition operations take streams of event messages as input and are fundamental components of all event stream processing systems. They were included in stream processing systems developed in research departments in the early part of the current century. For more information see the Aurora project [2] and Borealis project [1] from Brandeis, Brown and MIT, or the STREAM project [6] from Stanford. Further material on event processing can be found in [5].

Foundations

Transformations may be classified into the following types, depending on the nature of their input and output:

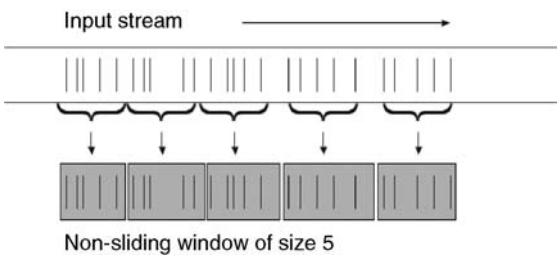
- *Translation.* Gets one event at a time, applies a translation operation and outputs the translated event. A translation operation is sometimes referred to as a *mapping*.
- *Split.* Gets one event at a time and splits the event to N different events, each of which contains a (possibly overlapping) subset of the event attributes.
- *Aggregation.* Takes a stream of events as input, and uses a windowing process to collect events from that stream into windows. The operation then processes each window to produce zero or more "derived" events as output.
- *Composition.* Takes two or more input streams and builds a window for each stream. The operation then takes this set of windows and processes them to produce zero or more derived events as output.

Aggregation and composition both operate against streams of events and since a stream is potentially unbounded they require the concept of a window defined over these input streams. A window is simply a set of event messages that is fetched from the input stream; the operation itself defines how the window is constructed from the stream and the conditions under which it is considered ready for processing. Examples include:

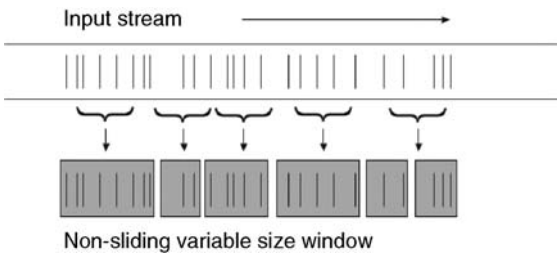
1. Non-sliding window of fixed size N. Events from the input stream are assigned to an "open window." When the specified number of events (N) has been received, the window is closed, ready to be

processed by the operation, and a new window is opened. Each input event is assigned to one and only one window (see Fig. 1).

2. Non-sliding variable size window. This is similar to the fixed-size non-sliding window, except that the closing of the window is triggered by some other condition, for example the detection of a change of type of the incoming event or the detection of a change in the value of a particular attribute on an incoming event (see Fig. 2).
3. Non-sliding time-based window. The operator specifies a time duration and a series of non-overlapping windows is created each of the specified duration. Events are assigned to a window instance based on some timestamp (for example arrival time) associated with the event. Such as if the window is specified to have a length of 2 s, then the window corresponding to time T would contain all events whose timestamp t lies between time T and T-2 seconds (see Fig. 3).
4. Sliding window of fixed size N. A new window is created when each incoming event is encountered. The incoming event and the preceding N-1 events (if any) are assigned to the window, which is then processed by the operation. Unlike the non-sliding examples, each event is assigned to multiple windows (see Fig. 4).



Event Transformation. Figure 1. Non-sliding window of fixed size.

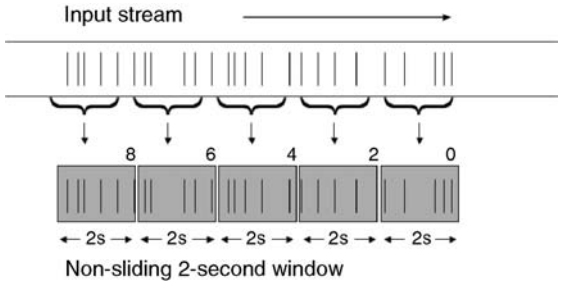


Event Transformation. Figure 2. Non-sliding variable size window.

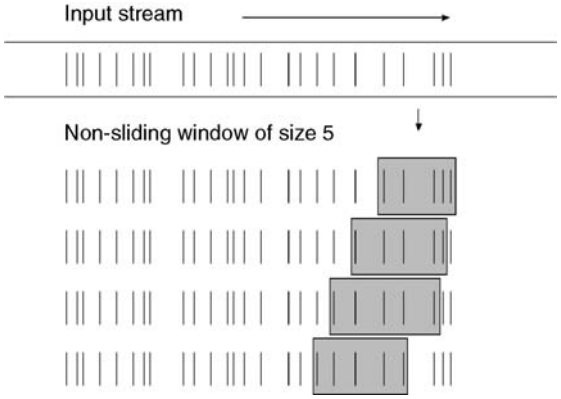
5. Sliding time-based window. A new window is created when each incoming event is encountered, but each window has the same fixed time duration. When a window is created, the incoming event is assigned to the window, along with all preceding events that lie within its time period, and the window is then processed. An alternative, possibly more natural, way of visualizing this is that there is a single window which moves in time; events are added to the window when they are detected, and removed from the window when they have overstayed the time limit. With either definition, if the window is specified to have a length of 2 s, then at time T the window would contain all events whose timestamp t lies between time T and T-2 s. (see Fig. 5).

Non-sliding windows are sometimes referred to as *tumbling* windows.

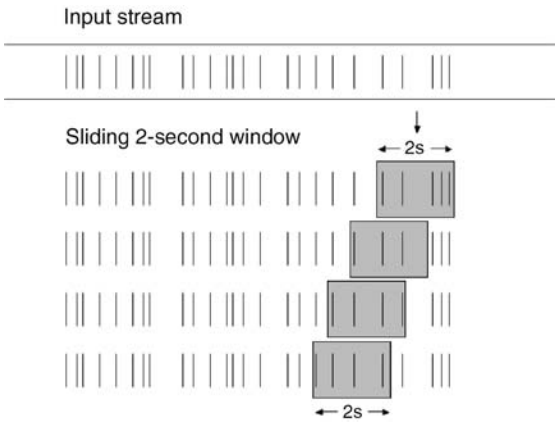
Translate and split style transformations have no requirement for a window, although they could be regarded as operating on a non-sliding window of size 1.



Event Transformation. Figure 3. Non-sliding time-based window.



Event Transformation. Figure 4. Sliding window of fixed size.



Event Transformation. Figure 5. Sliding time-based window.

A transformation where the result of processing a given event is affected by previous events from the input stream is referred to as a Stateful Transformation, whereas transformations where previous events have no effect on the processing of subsequent events are referred to as stateless transformations. Aggregation and composition are stateful by definition, whereas translate (and arguably split) can be either stateful or stateless.

Translation

A translation operation takes one event at a time and transforms it into another. Translation operations can be categorized into a number of different types depending on the amount of change they make to the incoming event.

The least intrusive type of event translation is one that changes the data representation (serialized form) of the event message, without changing its content. For example a translation might be used to convert between an XML and a Comma-Separated Value representation of the event, or to convert between different character sets.

Translation operations are frequently used to make more substantial changes to the content, and even the type, of the event:

- A translation might keep the content of the event unchanged, but modify the types used to represent the attributes of the event. For example it might convert a temperature attribute from Celsius to Fahrenheit, or replace a US state name with a two character code.
- A translation might modify the values of certain attributes of the event. This kind of translation can be used for data cleansing, for example replacing obviously incorrect date values.
- A translation can remove attributes from an event, while preserving the basic type structure of the event. Attributes could be removed if they are not needed by the event consumers, or because they contain sensitive information. A transformation that removes attributes is sometimes called a *projection*.
- Attributes can be added to the event, this is sometimes called *event enrichment*. The values assigned to the additional attributes can be:
  - Fixed values assigned by the translation, for example explicitly filling in default values for attributes omitted from the input event.
  - Values computed from other attributes in the event, for example the translation could assign a severity level to the event.
  - Values computed using an external source of information in addition to data from the event. An example of this would be a translation that uses a customer ID attribute from the input event to retrieve a customer name and address from a database and then adds these as attributes to the output event.

Translations are frequently programmed using visual editing tools, and can be represented using SQL queries, XML Style Sheets [3] that operate on an XML Infoset representation of the event, or other transformation languages.

Splitting

A splitting operation takes a single incoming event as input and produces multiple events as output. It is equivalent to taking a number of copies of the incoming event and applying a different translation to each. Splitting transformations are frequently used in combination with routing operations, with the different output events being routed to different recipients.

Some kinds of splitting transformation are particularly noteworthy:

- Splits that leave the original event untouched, but produce one or more additional events that are projections of the original event.
- Splits that break the original event apart by attribute, and produce a projected for each one of the attributes of the original event.

- Splits that iterate through lists in the original event and produce one projected event for each list item.

A split transformation can be programmed as a set of translations, this is sometimes referred to as a *static split*, since the number of output events is determined by the specification of the split operation. In contrast, in an *iterative split* the number of output events is determined by the data contained in the input event.

Aggregation

An aggregation operation is a function that operates on the events in a window to produce a single output event. The output event summarizes some aspect or aspects of the input events, by either:

- Concatenating the content of the input events, or
- Performing a function on the attributes of the input events; or

Examples of such functions include:

- Maximum, Minimum or "Top-k" values of an attribute.
- Sum or Mean values of an attribute.
- Other forms of average value of an attribute such as median or mode.
- Weighted averages, where one attribute in the event is the value to be averaged and another gives its weight. Weighted attributes are used to compute Volume-Weighted Average Price (VWAP) used in algorithmic trading applications.
- Count of the number of instances of an attribute, or the number of distinct values of an attribute.

An aggregation operation can also carry forwards a value from the previous window. For example when computing a "running maximum" of an attribute, or a VWAP, one may want to consider all events in a given time period (such as the current trading day), rather than just the events in the current window.

The choice of windowing style affects the output of the aggregation. For example an Averaging function with a non-sliding (tumbling) window will give a set of discrete averages, one for each window, whereas the use of sliding window will give a rolling average.

Composition

A composition operation takes windows from two (or more) input streams and performs an operation on them that produces zero, one or more output events. The most common type of composition operation is a *Join*.

Join operations are frequently used with sliding windows, to perform correlation between two streams. Each time an incoming event is detected on one of the streams, it is compared against the events in the window in the other stream using some kind of comparison test, and an event is output if a match is detected. This output event is produced by performing an operation on the attributes of the two matching events. In a two-way join operation both input streams are treated equally, and an incoming event on either stream is compared against the window of events on the other stream.

If the input streams are homogenous, all the events in a window have the same type, then standard relational Join operators can be used, with the two windows being treated as the two tables that are to be joined.

Key Applications

Event transformation of the types described here can be found in most event processing applications, for example in real-time monitoring, algorithmic trading, business activity monitoring.

Event translation has a particular use when integrating event producers and consumers that were not originally intended to work together.

Event aggregation and composition have a particular use in monitoring and simple pattern detection applications.

Cross-references

- ▶ Event Processing Network
- ▶ Event Routing
- ▶ Event Specification

Recommended Reading

1. Brandeis University, Brown University, and MIT, Borealis – Distributed Stream Processing Engine. http://www.cs.brown.edu/research/borealis/public/
2. Carney D. et al. Monitoring streams – A new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 215–226.
3. Clark J. (ed.). XML Transformations (XSLT) 1.0. W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116, 1999.
4. Hohpe G. and Woolf B. Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions. Addison-Wesley, Reading, MA, 2004.

5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Stanford University, STREAM project. http://infolab.stanford.edu/stream/

Event Uncertainty

► Uncertainty in Events

Event-Condition-Action Rules

► ECA Rules

Event-Driven Business Process Management

Rainer von Ammon
Center for Information Technology Transfer GmbH (CITT), Regensburg, Germany

Synonyms

Workflow management; Business process monitoring

Definition

The term *Event-Driven Business Process Management* is a combination of actually two different disciplines: Business Process Management (BPM) and Complex Event Processing (CEP). The common understanding behind BPM is that each company's unique way of doing business is captured in its business processes. For this reason, business processes are today seen as the most valuable corporate asset. In the context of this entry, BPM means a software platform which provides companies the ability to model, manage, and optimize these processes for significant gain. As an independent system, Complex Event Processing (CEP) is a parallel running platform that analyses and processes events. The BPM- and the CEP-platform correspond via events which are produced by the BPM-workflow engine and by the IT services which are associated with the business process steps.

Historical Background

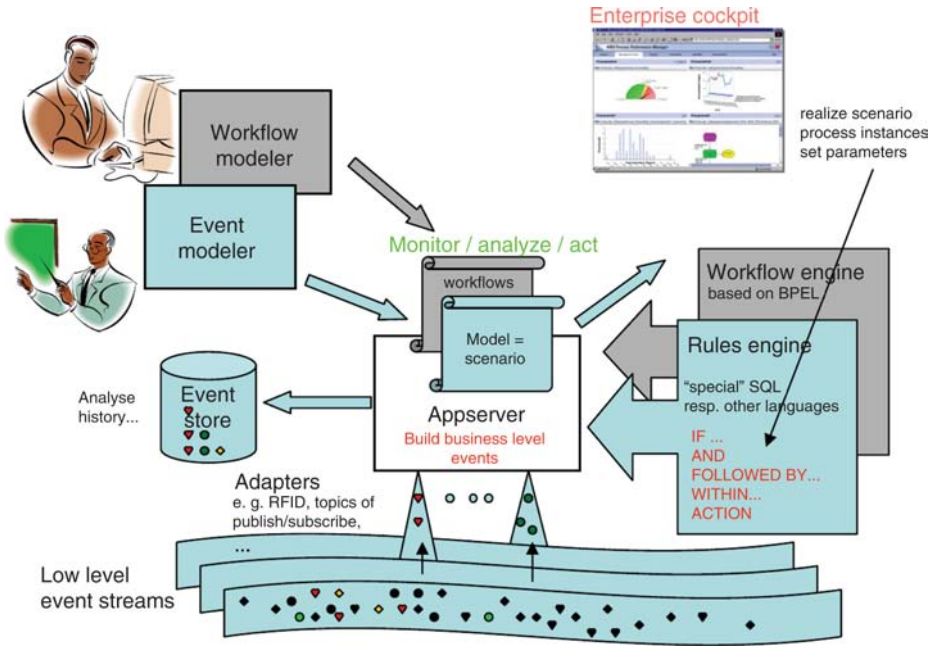
The term "Event Driven Business Process Management" was first used in June 2003 in a white paper of Bruce Silver Associates in connection with the FileNet P8-BPM platform [3]. The term was understood as a synthesis of workflow and Enterprise Application Integration (EAI). Actually, also a concept of event processing and real-time BAM was described in that white paper already, but only as single event processing without knowing anything about CEP which was not a discipline yet. Although the book "The power of events" of David Luckham [7] was already published in 2002, it wasn't well known until 2004 and there was no really working CEP platform so far. In November 2007, the term was explicitly used as a combination of BPM and CEP in an interview with Ruma Sanyal from BEA Systems [4].

Foundations

Figure 1 shows the principle of how a BPM- and BAM/CEP platform work together on the basis of events. The dark shaded components concern workflow respectively BPM, the light shaded components concern the real-time BAM/CEP components. Just started with the availability of commercial CEP platforms, there will be two different kinds of specialists in the future: the workflow modelers or business analysts and the event modelers.

The Business Process Modeler

The workflow modeler identifies business processes, starting from the value chain of an enterprise, he analyses and reengineers or optimizes the business processes by using a Business Process Analyzing-(BPA) tool based on a standard notation like the Business Process Modeling Notation (BPMN), invented 2002 by IBM and now standardized by the Object Management Group (OMG) since 2005, published February 2006 as version 1.0 [8]. A business process model is a graphically designed network based on symbols e.g., for flow objects like activities, gateways and also events. Depending on their position in the network, BPMN categorizes the events as start events, intermediate events and stop events. Each event category has different event types like timer-, message-, exception events etc. In the present release of BPMN 1.1, there are around 25 event types, the new release BPMN 2.0 is announced for Q3/2008. The modeling of business processes is a highly skilled task for a workflow modeler respectively a business analyst, but



Event-Driven Business Process Management. Figure 1. A reference model of the adoption of a CEP/BAM/BPM platform [10].

completely different from the task of an event modeler for real-time BAM/CEP. The event modeler acts with different kinds of events which are produced by the business process instances or from other event sources, like SNMP traps, RFID, log file entries etc.

[The business process models can be transformed into an executable notation of a workflow engine, respectively a BPM platform.] Executable notations are being standardized by OASIS as Business Process Execution Language (BPEL) or as a different standard XPDL by the Workflow Management Coalition (WfMC). To make a business process model really executable, it has to be remodeled more fine grained directly in the BPM platform. From the point of view of the IT department, for example, so called compensations and exceptions have to be modeled, if an IT-service cannot be executed or fails. This task cannot be done by the business analyst who does not know anything about the internals of IT services associated with the process steps. So, in the future as well as with the upcoming BPM platforms and the standards a new procedure for modeling and implementing business processes is needed. Business analysts from the operating departments must closely work with IT specialists, modeling the process and its associated IT services together. This is visualized as BPMN, but directly in an integrated modeling tool of the BPM platform. This is enhanced by a user interface for defining technical information (such as port-types, ports, partnerlinks, peoplelinks etc.), which is needed for the process execution, e.g., by a BPEL-engine. The upcoming BPM platforms are also able to produce events automatically when a service is called or cannot be executed and fails and so on. These are the event types the event modeler has to know for realizing BAM dashboards and enterprise cockpits.

The Event Modeler

By cooperating with the process owners of the operating departments or even with the C-level management of an enterprise, the event modeler has to define which BAM view has to be monitored in a dashboard, which alerts are to send to which roles in the organization and which actions shall be started automatically if a certain event pattern occurs. Derived from such BAM views, the event modeler looks for the needed event types and their instances flowing through the event streams of an enterprise or that are saved in an event store. It requires a high level of skill to define the right event patterns for a real-time BAM view. The event modeler has to know the different event sources like JMS messages realized on the basis of Publish/Subscribe topics etc. He has to install the corresponding event adapters delivered by the

CEP platform as "out of the box" prebuilt features or the event modeler has to care about the development of not yet existing adapters.

Event Processing Languages

The event modeler defines the event patterns for a BAM view on the basis of an Event Processing Language (EPL). At present, there is no standard for an EPL. The CEP community, founded as a discipline at the first CEP symposium in Hawthorne/New York in March 2006, is discussing the right standard for an EPL. It seems that there will be different EPL-approaches for different domains like Algorithmic Trading, BAM etc. At present, the CEP community is gathering use cases and is classifying them according to their corresponding domains [5]. The CEP platforms come with an SQL-like language (e.g., Coral8, Esper, Oracle, StreamBase) or provide a rule-based EPL approach (e.g., AMiT from IBM) or have an abstract user interface that hides any language and generates code like Java (e.g., Tibco, AptSoft).

The models of business processes and event scenarios are deployed into a middleware platform, e.g., into an application server, which is responsible for high availability, limitless scalability, grid computing, failover, transparency of heterogeneous infrastructures and so on.

Relation to the Database Technology

In connection with the term "Event-Driven BPM", there are two major challenges from the point of view of database technologies.

The challenge from the BPM side is storing millions of instances of long running business processes and their status. An enterprise such as a bank or the automotive industry has several thousand business processes like different types of an account opening for a depot, giro account, credit application, project budget confirmation etc. Each business process has several thousand process instances like the credit application of Betty Miller, John Smith etc. For long running processes, the BPM platform has to keep the status of the process instances, e.g., when waiting for the signature of Mrs. Miller or for the confirmation of the credit conditions by the bank. Because that can take days or weeks, there are millions of active business processes in a certain time window at the same time. That is the reason, for example, for overflows of queues in current projects.

The challenge from the BAM/CEP side is processing thousands of events per second generated by business processes and their associated IT-services by in-memory databases, realizing time windows. A memory-centric SQL engine is designed to execute queries without ever writing data to disk. Such an engine also provides a persistence feature for storing the data as "historic events" in an event store for later processing, i.e., for aggregating and correlating them with the current events of the event streams. The basic idea is to route an event through a lot of different in-memory filters, to see what queries it satisfies, rather than executing many queries in sequence against disk-based data. That is a precondition for realizing real-time BAM or the idea of "predictive business" [9].

Examples

If an online credit system receives 50,000 or 100,000 credit applications per day, the bank likes to monitor the application process by typical event patterns that signal, for example, that a credit application could be cancelled within the next moment. In this case, the system shall trigger an action to keep the customer. For this purpose, the event modeler has to identify the corresponding events. If not available from the implemented process, he has to take care of the generation of additional appropriate events. This is a demanding, challenging task.

As another example, BAM/CEP shall send an alert to an employee of the bank, because the loss by cancellations exceeds a certain threshold. However, it wouldn't make sense to send an alert each time a cancellation happens, because too many alerts may result, and the VIRT problem (valuable information at the right time [6]) arises.

Key Applications

Key applications will take place in all domains and first projects on the basis of *Event Driven BPM*-platforms just start 2008:

Logistics applications, e.g., at DHL/Deutsche Post [5].

Finance applications, e.g., Deutsche Bank, TeamBank [2].

Telco applications, e.g., Deutsche Telekom, T-Mobile.

Current/future research projects, e.g., «Domain specific reference models for event patterns for a faster set-up of BPM/BAM applications» [1].

Cross-references

- ▶ Business Activity Monitoring
- ▶ Complex Event Processing
- ▶ Event Driven Architecture
- ▶ Event Stream Processing
- ▶ Service Oriented Architecture

Recommended Reading

1. Ammon R.V., Silberbauer C., and Wolff C. Domain specific reference models for event patterns – for faster developing of business activity monitoring applications. In Proc. VIP Symp. on Internet Related Research, 2007.
2. Brandl H.-M. and Guschakowski D. Complex Event Processing in the Context of Business Activity Monitoring. An Evaluation of Different Approaches and Tools Taking the Example of the Next Generation Easycredit. Diploma thesis, 2007. http://www.citt-online.de/downloads/Diplomarbeit_BaGu_Final.pdf
3. Bruce Silver Associates. FileNet P8 – event-driven business process management. Industry trend reports, June 2003. Available at: http://www.ecm-unverzichtbar.de/Repository/48/files/69_EvBPM.pdf; http://www.complexevents.com/
4. Danielsson K. and Trotta G. Key requirements for event-driven BPM and SOA. Available at: http://www.ebizq.net/topics/bpm/features/8700.html
5. Emmersberger C. and Springer F. Event Driven Business Process Management. An Evaluation of the Approach of Oracle and the Sopera Open Source Framework. Diploma thesis, 2008. http://www.citt-online.de/downloads/EmmSpr_Diplomarbeit_Final.pdf
6. Hayes-Roth F. Value information at the right time (VIRT): Why less volume is more value in hastily formed networks. Available at: www.nps.edu/cebrowski/Docs/VIRTforHFNs.pdf, 2007.
7. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
8. OMG/BPMI. Business process modeling notation (BPMN) information. Available at: http://www.bpmn.org/
9. Ranadive V. The Power to Predict: How Real Time Businesses Anticipate Customer Needs, Create Opportunities and Beat the Competition. McGraw-Hill, New York, NY, 2006.
10. Use cases of CEP applications. Available at: http://complexevents.com/?cat=16

Eventual Consistency

Marc Shapiro[1], Bettina Kemme[2]
[1]INRIA Paris-Rocquencourt and LIP6, Paris, France
[2]McGill University, Montreal, QC, Canada

Definition

In a replicated database, the consistency level defines whether and how the values of the replicas of a logical object may diverge in the presence of updates. Eventual consistency is the weakest consistency level that guarantees useful properties. Informally, it requires that all replicas of an object will eventually reach the same, correct, final value, assuming that no new updates are submitted to the object.

Key Points

Eventual consistency is an important correctness criterion in systems with a lazy, update-anywhere strategy, also referred to as *optimistic replication* (q.v.). Update operations can be submitted and executed on any node, and the propagation of updates occurs lazily after commit. Conflict resolution and reconciliation must ensure that all replicas (copies) of a logical object eventually converge to the same value. Different objects are considered independent.

In a system where updates are continuously submitted, eventual consistency can be defined by a weak form of schedule equivalence [1]. A schedule S_n^x describes the sequence of update operations node n performs on its replica of object x. An element of the schedule of the form w_i represents the execution of an update to object x, submitted by some user. S_n^x contains an element of the form $\overline{w_i}$, if the update w_i was received by n, but either not executed, or aborted due to conflict resolution.

Typically, two schedules are defined equivalent by restricting how the order of operations in the two schedules may differ. However, for eventual consistency only the convergence of object values matters. Thus, equivalence is defined by comparing the final state of the replicas. Two schedules are said *state equivalent* when, starting from the same initial state, they produce the same final state. For instance: (i) schedules $S = w_1 w_2$ and $S' = w_2 w_1$ are state-equivalent if w_1 and w_2 commute; (ii) schedules $S = w_1 w_2$ and $S' = w_2$ are state-equivalent if w_2 sets the state of the object to a completely new value (e.g., $x := 2$).

Eventual consistency of a replicated object x is defined by the following conditions, which must hold at all times, at any node n with a replica of x [1]. It is assumed that all replicas have the same initial state. There is a prefix of the schedule S_n^x that contains the same operations and is state-equivalent to a prefix of the schedule $S_{n'}^x$ of any other node n' holding a replica of x. Such a prefix is called a *committed prefix* of S_n^x.

- The committed prefix of S_n^x grows monotonically over time, i.e., the extant set of operations and their relative order remains unchanged.
- For every operation w_i submitted by a user, either w_i or $\overline{w_i}$ eventually appears in the committed prefix of S_n^x (but not both, and not more than once).
- An operation of the form w_i in the committed prefix satisfies all its preconditions (e.g., the state of the object immediately before the execution of the operation fulfills certain conditions).

As an example, assume operation w_1 sets x to 2, and w_2 sets it to 5. Operation w_1 is submitted and executed at n_1 while w_2 is first executed at n_2. At this time the local schedules are $S_1 = w_1$ and $S_2 = w_2$ and the committed prefix at both nodes is the empty schedule. Now w_1 is propagated to n_2 and w_2 is propagated to n_1. When n_1 receives w_2, it detects that w_1 and w_2 are concurrent and conflict. Say that conflict reconciliation prioritizes one of the operations, e.g., w_1. Then, w_2 is simply not executed and the new schedule is $S_1^x = w_1\overline{w_2}$. At n_2, when w_1 arrives, the conflict is also detected, w_2 is undone, w_1 is executed and the final schedule is $S_2 = \overline{w_2}w_1$. At this time, S_1 and S_2 are themselves the committed prefixes. Note that further concurrent operations on x might move the schedules further, but the extensions would still be tentative and only become committed once they are reconciled at all replicas.

Cross-references

- ▶ Concurrency Control – Traditional Approaches
- ▶ Data Replication
- ▶ Distributed Concurrency Control
- ▶ Distributed database systems
- ▶ Middleware Support for Database Replication and Caching
- ▶ One-Copy-Serializability
- ▶ Optimistic Replication and Resolution
- ▶ Replica freshness
- ▶ Replicated Database Concurrency Control
- ▶ Strong Consistency Models for Replicated Data
- ▶ WAN Data Replication
- ▶ Weak Consistency Models for Replicated Data

Recommended Reading

1. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.

Evidence Based Medicine

Michael Weiner
Indiana University School of Medicine, Indianapolis, IN, USA

Synonyms

Evidence based practice; Scientific medicine

Definition

Evidence based medicine (EBM) is the "conscientious, explicit, and judicious use of current best evidence in making decisions about the care of individual patients" [1].

Key Points

Although medical practice is historically based on observation and best attempts at scientific method, early medical practices often involved trial and error or observations of one or a few cases. In the last century, advances in research, experience, technology, and communications have moved medicine more towards practice based on established evidence.

Individuals especially responsible for promoting, developing, describing, and discussing EBM have included Archie Cochrane, David Sackett [4], and Gordon Guyatt [2]. The international Cochrane Collaboration [3] has emerged to continue to advance EBM throughout the world. This and several other organizations, such as the US Preventive Services Task Force [1], have assembled, analyzed, and graded collections of medical evidence. Specific evidence reported from such groups often has gradings that indicate the strength of evidence about a topic, based on quality or scope of methods, statistics, reproducibility of findings, etc.

The representation of medical evidence in information systems or databases can assume various forms, especially because the type of evidence is likely to vary substantially from one topic to the next. Nevertheless, the use of standard clinical research methods, such as randomized controlled trials, survival analysis, and meta-analysis, can facilitate knowledge management and representation by limiting the types of information that need to be stored or communicated. Automated extraction of medical knowledge from research reports may benefit from natural language processing when reports are not more structured.

EBM has several positive aspects. It can help to increase adherence to clinical guidelines and minimize variations in medical practice that may be viewed as too large or experimental. It can help to reduce disparities and establish a minimum standard of care for a population. To some degree, EBM seems to hold practitioners and researchers more accountable for their actions and pursuits and also distributes the knowledge among a larger group of professionals and consumers. EBM promotes the development of further research.

Overreliance on EBM may stifle creativity, or it may fail to take into account an individual's preferences or characteristics that may not "fit the mold" or may require special handling not covered by a guideline or established evidence. Thus, practicing EBM requires the ability to find, understand, interpret, and appropriately apply evidence with individual patients or groups. The Internet is increasingly a useful source of EBM knowledge, via databases, literature reviews, and communications.

The rise of EBM has been associated with corresponding local and federal health policies. For example, medical institutions may prohibit certain procedures without adequate evidence of benefit. Not only the practice of medicine, but the reimbursement of modern medical care, often depends on evidence.

Cross-references

- ▶ Classification
- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Indexing of Data Warehouses
- ▶ Information Extraction
- ▶ Storage Management
- ▶ Text Mining
- ▶ Visual Content Analysis
- ▶ Web Information Extraction

Recommended Reading

1. Agency for Healthcare Research and Quality. U.S. Preventive Services Task Force (USPSTF). 2007. Available online at http://www.ahcpr.gov/clinic/uspstfix.htm (accessed Aug 30, 2007).
2. Evidence-Based Medicine Working Group. Evidence-based medicine. A new approach to teaching the practice of medicine. JAMA. 268(17):2420–2425, 1992.
3. The Cochrane Collaboration. The Cochrane Collaboration. 2007. Available online at http://www.cochrane.org/ (accessed Aug 30, 2007).
4. Sackett D.L., Rosenberg W.M., Gray J.A., Haynes R.B., and Richardson W.S. Evidence based medicine: what it is and what it isn't. BMJ. 312(7023):71–72, 1996.

Evidence Based Practice

▶ Evidence Based Medicine

Evolutionary Algorithms

▶ Genetic Algorithms

Evolutionary Computation

▶ Genetic Algorithms

Evolutionary Semantics

▶ Emergent Semantics

Exactly Once Execution

▶ Application Recovery

Executable Knowledge

Mor Peleg
University of Haifa, Haifa, Israel

Synonyms

Computer-interpretable formalism; Knowledge-based systems

Definition

Executable knowledge is represented in a symbolic formalism that can be understood by human beings and interpreted and executed by a computer program. It allows a computer program to match case data to the knowledge, reason with the knowledge, select recommended actions that are specific to the case data, and deliver them to users. Executed knowledge can be delivered in the form of advice, alerts, and reminders, and can be used in decision-support or process management.

Historical Background

Representing knowledge in a computer-interpretable format and reasoning with it so as to support humans in decision making started to be developed by the artificial intelligence community in the 1970s. According to Newell [6], knowledge is separate from its representation. At the knowledge level, an agent has as parts bodies of knowledge, actions, and goals. An agent processes its knowledge to determine the actions to take. An agent, behaving through the principal of rationality, selects those actions that attain his goals. Representing knowledge in a symbolic way (i.e., data structures and algorithms) allows the expression of knowledge in a way that would be understandable by humans and would allow people and computer programs to reason with the knowledge and select actions in the service of goals. According to Davis et al. [1], knowledge representation serves five roles: (i) a surrogate to enable an entity to determine the consequences of thinking, (ii) a set of ontological commitments about how and what to see in the world, (iii) a fragmentary theory of intelligent reasoning, (iv) a medium for efficient computation, and (v) a medium for human expression.

Research on knowledge representations started to evolve in the 1970s with rule-based systems. One of the most famous rule-based systems is Mycin [11]. This system represented clinical knowledge about diagnosis and treatment of infectious diseases and allowed clinician users to enter patient findings and, through execution of the if-then-else rules, arrive at probable diagnoses and appropriate treatments. In the 1980s decision-theoretic models, such as Bayesian Networks, influence diagrams, and decision trees [7] started to become popular ways of representing and reasoning with decision knowledge in a probabilistic way, under uncertainty.

By the late 1980s, researchers started to realize the importance of integrating knowledge-based systems with databases. In this way, the case data coming from a database could drive the execution of encoded knowledge. An example of an early formalism that addressed data integration is the Arden Syntax for Medical Logic Modules (MLMs) [4], discussed in detail below.

In the 1990, ontologies [3] were developed to formalize a shared understanding of a domain. In knowledge engineering, the term ontology is used to mean definitions of concepts in a domain of interest and the relationships among them ("a specification of a conceptualization of a domain" [3]). An ontology enables software applications and humans to share and reuse the knowledge consistently. Ontologies, as represented in a formal language such as frames or description logic, allow logical inference over the set of concepts and relationships to provide decision support and explanation facilities. Ontologies are much more maintainable than rule-based systems. In rule-based systems, the knowledge is represented as individual rules. It is difficult to foresee the affect of addition, deletion, or modification of a rule on the performance of the rule-based system. This problem is solved by ontologies. Research in the 1990s, led by Musen [5], addressed modeling of domain knowledge and problem-solving methods as ontologies that could be combined together, such that a problem solving method could be applied to several domain ontologies, and decision-support systems in a single domain could utilize several generic problem-solving methods.

In the 1990s, a new generation of executable knowledge-based systems started to be developed by two separate communities: the medical informatics community and the business process community. The common theme to the new developments by both of these communities was that the executable knowledge was no longer representing individual decisions, but rather a process that unfolds over time and includes many activities, and the fact that the knowledge-based process system had to be integrated with other systems in the organization. The process modeling languages in both communities were developed as ontologies.

Foundations

Executable knowledge formalisms were developed by several communities, including the artificial intelligence community, the medical informatics community, and the business process management community. The rest of this article reviews work done by the medical informatics and business process communities.

Following the success of rule-based decision-support systems, and at the same time, recognizing the advantages of linking a knowledge-based system to case data, a standard for encoding individual medical decisions, known as the Arden Syntax for Medical Logic Modules (MLMs) [4], was defined in 1989. Arden Syntax was developed initially under the sponsorship of the American Society for Testing and

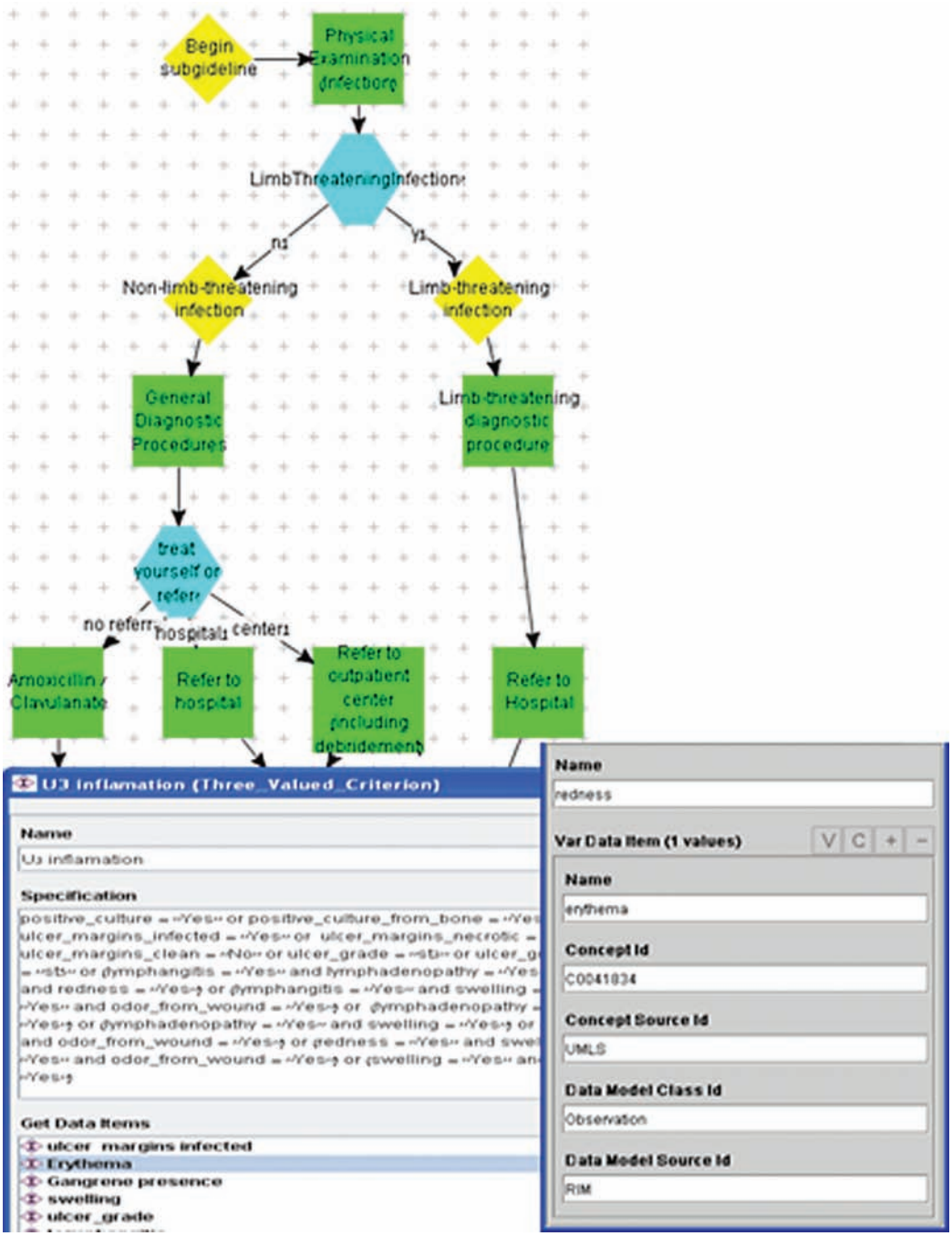
Materials and subsequently of Health Level Seven (HL7). Arden Syntax is a published as an American National Standards Institute (ANSI) standard. MLMs, in Arden Syntax, define decision logic via a knowledge category that has data, evoke, logic, and action slots. The event, logic and action slots specify respectively the events that trigger the MLM (e.g., storage of a new serum potassium test result into a database), the logical criterion that is evaluated (e.g., serum potassium value <3.5), and the action that is performed if the logical criterion holds, which is often an alert or reminder (e.g., potassium replacement therapy). The data slot specifies mappings between the specific database records and the MLM's variables. However, only part of this specification is defined by the syntax, as it does not contain standard terminology or a data model for electronic medical records.

In the 1990s, decision-support systems whose knowledge is based on evidence-based clinical guidelines started to be developed. Clinical guidelines are recommendations that are developed by healthcare organizations based on evidence from clinical trials, and are aimed at assisting practitioner and patient decisions about appropriate healthcare for specific clinical circumstances. Unlike individual clinical decisions, such as MLMs, clinical guidelines involve multi-step decisions and actions that unfold over time. Guideline-based decision-support systems help clinicians in the process of patient care, including decision making and task management. These systems are based on process-flow ontologies termed Task-Network Models (TNM) [10] – a hierarchical decomposition of guidelines into networks of component tasks that unfold over time. The task types vary in different TNMs, yet all of them support modeling of medical actions, decisions, and nested tasks. These models contain computer-interpretable specifications of decision criteria and clinical actions that enable an execution engine to interpret the guideline representation and execute it for a given patient case data. Important themes in executable guideline models include the ability of some of them to integrate with electronic databases (e.g., electronic health record systems (EHR)), using standard terminologies to express medical actions (e.g., laboratory tests, drug prescriptions) and patient data items upon which decision criteria are written, using standard expression languages for writing decision criteria, and using messaging standards for exchange of (clinical) data.

In 2003, researchers from six groups that developed TNMs participated in a study that compared their models using two clinical guidelines that served as case studies [10]. The TNMs studied were Asbru, EON, GLIF, Guide, PRODIGY, and PRO*forma*. An example of a guideline model represented in GLIF is shown in Fig. 1. Although these formalisms all depict a guideline as a TNM, they each have their own emphasis. Asbru emphasizes specifying intentions as temporal patterns; EON views the guideline model as the core of an extensible set of models, such as a model for performing temporal abstractions. EON uses a task-based approach to define decision-support services that can be implemented using alternative techniques; GLIF emphasizes the ability to share and integrate guideline specifications among software tools and implementing institutions; GUIDE focuses on integration with organizational workflow using a workflow-based model and linkage to decision-theoretic models, PRODIGY (a project that is no longer active) aimed at producing the simplest, most readily comprehensible model necessary to represent chronic disease management guidelines. It models guidelines as decision maps organized as a collection of clinical contexts; in each context, selection among relevant clinical actions is made; PRO*forma* advocates the support of safe guideline-based decision support and patient management by combining logic programming and object-oriented modeling. Its syntax and semantics are formally defined. One aim of the PRO*forma* project is to explore the expressiveness of a deliberately minimal set of modeling constructs: actions, compound plans, decisions, and inquiries of patient data from a user.

The study compared the TNM models in term of eight components that capture the structure of computerized guidelines: (i) organization of guideline plans, (ii) goals, (iii) model of guideline actions, (iv) decision model, (v) expression language, (vi) data interpretation/abstractions, (vii) medical concept model, and (viii) patient information model.

The purpose of the study was to find consensus among the different formalisms that could be a starting point for creating a standard computerized guideline formalism. Differences between the guideline modeling languages were most apparent in underlying decision models (ranging from simple switching constructs to argumentation rules for and against decision alternatives, and even use of decision-theoretic models such as influence diagrams and decision trees), goal



Executable Knowledge. Figure 1. Part of a GLIF model for a Diabetic Foot guideline, modeled using the Protégé-200 tool. Squares denote actions, diamonds – patient state steps, hexagons – decisions. The bottom part of the figure shows the computable specification of the decision step "Limb-threatening Infection." The decision criterion is based on values of patient data items. Each patient data item is defined using a concept code taken from a controlled medical vocabulary (Unified Medical Language System (UMLS) in this case) and by a patient data model class (e.g., Observation) taken from a data model source, such as Health Level 7's Reference Information Model (RIM).

representation, use of scenarios (as a plan component that defines a particular patient management context that serves as entry points into guidelines), and structured medical actions which could be mapped into controlled vocabulary terms.

Consensus was found in plan organization (plans could be nested and structured as plan components arranged in sequence, in parallel, and in iterative and cyclic structures), expression language for specifying and sharing decision and eligibility criteria, patient state definitions, and preconditions on system actions, conceptual medical record model, medical concept model, and data abstractions (i.e., definitions of abstract terms using mathematical functions of other concepts, temporal abstractions, and concept hierarchies that allow reasoning at different levels of the hierarchy).

The HL7 Clinical Decision Support Technical Committee (CDSTC) has focused on standardization of two of the components for which consensus was found: expression language and conceptual virtual medical record (vMR) model. The object-oriented guideline expression language, GELLO [12], is an extensible guideline expression language that can be used for formally defining decision and eligibility criteria, as well as patient states. It is based on the Object Constraint Language (OCL) (http://www-306.ibm.com/software/rational/uml/resources/documentation.html). In 2004, it was established as a standard of HL7. The CDSTC started the process of standardizing a vMR, based on experiences with the patient information models of PRODIGY, EON and the HL7 RIM, which is also the basis of GLIF's default patient information model. An object oriented vMR would ease the process of mapping guideline patient data items to real EMRs, allowing decision criteria, eligibility criteria and patient states to be defined by in guideline models by reference to the VMR rather than specific EMRs.

The second community that started to develop a new generation of executable knowledge-based systems is the business process management community. Workflow management systems started to be developed based on workflow formalisms that define a model of business processes and a model of the organization, including its individual actors, roles, organizational units, and resources that participate in workflow activities. Several industrial groups defined standards for workflows. The Web Services Business Process Execution Language (BPEL) by the Advanced Open Standards for the Information Society (OASIS), is an executable formalism where the business process behavior is based on Web Services. XML Process Definition Language (XPDL) is a standard that is under development since 1998 by the Workflow Management Coalition – An industry group dedicated to creating software standards for workflow applications. The goal of XPDL is to store and exchange the process diagrams among different workflow tools, including workflow modeling tools (editors) and workflow engines. In 2004 the Workflow Management coalition endorsed the graphical workflow standard called Business Process Modeling Notation (BPMN) developed by Object Management Group, to standardize the way that process definitions are visualized.

The business process management community has developed many tools that enable modeling, analyzing, verifying, simulating, and executing a business process. While the medical informatics community devoted much research to supporting clinical decision making in addition to patient care task management, the business process community made much progress on modeling task management within an organization, where the business process often involves many departmental units and organizational roles. Workflow systems aim to help in resource management, a task that is not addressed by current executable guideline systems.

Key Applications

A generic free and open-source tool for creating ontologies, or knowledge-bases, is Protégé (http://protege.stanford.edu). The Protégé platform supports two main ways of modeling ontologies via the frames and Ontology Web Language (OWL) formalisms. Protégé is implemented in the Java programming language and is extensible. Protégé-frames knowledge bases could be reasoned with several rule-engines, such as Jess and Algernon. Additionally, the Protégé Axiom Language could be used to define first-order-logic constraints and check them for instances in the knowledge base. Several reasoners can be used with Protégé-OWL to classify instances or classes according to class definitions and class hierarchies. They include the commercial product Racer, and the commonly used free reasoners Pellet and Fact + +. The SWRL rule-based engine could also be used to reason with Protege-OWL.

The Arden Syntax for Medical Logic Modules has been applied by universities such as Columbia

University Medical School and by several health information systems vendors, such as Micromedex, Eclypsis Corporation, McKesson Information Solutions, and Siemens Medical Solutions Health Services Corporation to create MLMs that deliver clinical recommendations in the form of alerts and reminders. MLMs are being used by many clinical institutions in the United States (see http://cslxinfmtcs.csmc.edu/hl7/arden/ for a partial list). The Department of BioMedical Informatics at Columbia University developed a knowledge-base of over 240 MLMs that are now available from http://cslxinfmtcs.csmc.edu/hl7/arden/.

Many tools exist to support guideline development, modeling, verification, and execution [2,8]. Asbru modeling tools include Delt/A (http://ieg.ifs.tuwien.ac.at/projects/delta/) and URUZ, both focusing on easing the transition from narrative to formal representations via a mark-up stage, AsbruView (http://www.ifs.tuwien.ac.at/asgaard/asbru/tools.html), which focuses on visualization and user interface for authoring, and CareVis (http://ieg.ifs.tuwien.ac.at/projects/carevis/), which provides multiple simultaneous views to cover different aspects of a complex underlying data structure of treatment plans and patient data. Verification of Asbru guidelines can be done using formal verification methods [13]. Implementations in Asbru were developed for diabetes, artificial ventilation, and breast cancer guidelines.

EON guidelines can be authored in the knowledge-modeling tool Protégé-2000 and executed by an execution engine that uses a temporal data mediator to support queries involving temporal abstractions and temporal relationships. A third component provides explanation services for other components. EON has been used to create hypertension and opiods guidelines that are implemented in various hospitals and clinics of the Veteran Affairs Hospital. Protégé-2000 is also the modeling tool for GLIF guidelines. The GLIF execution engine (GLEE) has been used to implement two guidelines: diabetic foot diagnosis and management and flu vaccinations.

Guide has a new implementation of an authoring tool and an execution engine called NewGuide. Guide has been used to implement guidelines for stroke and for management of patients with heart failure.

A number of software tools have been created for creating, visualizing, and executing PRO*forma* guidelines. They include Arezzo and Tallis. Several PRO*forma* guidelines have been implemented and some have undergone clinical trials to establish their safety and utility. More information on PRO*forma* implementations as well as on implementations of other guideline formalisms can be found at the open clinical web site (www.openclinical.org) – a repository of resources about decision support, clinical workflow and other advanced knowledge management technologies for patient care and clinical research.

Key application for workflow management systems include tools for modeling and executing workflows, such as the open-source workflow tool Bonita (http://bonita.objectweb.org), FLOWer (http://www.workflow-download.com/workflow/flower.html), YAWL (http://www.yawl-system.com/), and Together Workflow Editor andserver(http://www.together.at/together/prod/twe/), and Oracle Workflow (http://www.oracle.com/technology/products/integration/workflow/workflow_fov.html). Other tools exist for verifying [15] workflows (Oracle Worklow). There are business process management engines available from several vendors including Software AG, Savvion, Lombardi, Appian, JBoss, and Tibco.

Future Directions

The challenge of creating executable knowledge that can be shared by multiple implementing institutions and mapped to their information systems started to be addressed by projects such as the Arden Syntax, GLIF, Shareable Active Guideline Environment (SAGE) [14] and Knowledge-Data Ontology Mapper [9], yet more work needs to be done in this area to define how knowledge can be authored in a way that is institution-specific and sharable.

One of the most interesting future directions of executable knowledge concerns synergetic development that draws upon developments made in the medical informatics and the business process communities. Such collaborations are emerging, as manifested by health-care related workshops that are taking place in business-process management and information systems conferences, such as the ProHealth Workshop, which is part of the Business Process Management conference.

Cross-references

- ▶ Bayesian Classification
- ▶ Ontologics
- ▶ OWL: Web Ontology Language
- ▶ Rule-Based classification

Recommended Reading

1. Davis R., Shrobe H., and Szolovits P. What is a knowledge representation? AI Magazine, 14(1):17–33, 1993.
2. de Clercq P.A., Blom J.A., Korsten H.H.M., and Hasman A. Approaches for creating computer-interpretable guidelines that facilitate decision support. Artif. Intell. Med., 31:1–27, 2004.
3. Gruber T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Human Comput. Stud., 43:907–928, 1995.
4. Hripcsak G., Ludemann P., Pryor T.A., Wigertz O.B., and Clayton P.D. Rationale for the arden syntax. Comput. Biomed. Res., 27(4):291–324, 1994.
5. Musen M.A., Tu S.W., Eriksson H., Gennari J.H., and Puerta A.R. PROTEGE-II: an environment for reusable problem-solving methods and domain ontologies. In Proc. 13th Int. Joint Conf. on AI, 1993.
6. Newell A. The knowledge level. AI Magazine, 2(2):1–20, 33, 1980.
7. Pearl J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco, CA, 1988.
8. Peleg M. Guideline and workflow models. In Clinical Decision Support – The Road Ahead, R.A. Greenes (ed.). Elsevier/Academic Press, Orlando, FL, 2006.
9. Peleg M., Keren S., and Denekamp Y. Mapping computerized clinical guidelines to electronic medical records: knowledge-data ontological mapper (KDOM). J. Biomed. Inform., 41(1):180–201, 2008.
10. Peleg M., Tu S.W., Bury J., Ciccarese P., Fox J., Greenes R.A., et al. Comparing computer-interpretable guideline models: a case-study approach. J. Am. Med. Inform. Assoc., 10(1):52–68, 2003.
11. Shortliffe E.H. Computer-Based Medical Consultations: Mycin. Elsevier/North Holland, New York, 1976.
12. Sordo M., Ogunyemi O., Boxwala A.A., Greenes R.A., and Tu S. GELLO: a common expression language. Available online at: http://cslxinfmtcs.csmc.edu/hl7/arden/2004–09-ATL/v3ballot_gello_aug2004.zip, 2004.
13. ten Teije A., Marcos M., Balser M., van Croonenborg J., Duelli C., van Harmelen F., et al. Improving medical protocols by formal methods. Artif. Intell. Med., 36(3):193–209, 2006.
14. Tu S.W., Campbell J.R., Glasgow J., Nyman M.A., McClure R., McClay J.P.C., Hrabak K.M., Berg D., Weida T., Mansfield J.G., Musen M.A., and Abarbanel R.M. The SAGE guideline model: achievements and overview. J. Am. Med. Inform. Assoc., 14(5):589–598, 2007.
15. van der Aalst W.M.P. The application of petri nets to workflow management. J. Circuits Syst. Comput., 8(1):21–66, 1998.

Execution Skew

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Definition

Execution skew is a phenomenon observed during the parallel evaluation of a database query, in which the concurrent operators exhibit disparate execution times.

Key Points

Execution skew is a phenomenon observed in the parallel evaluation of a database query. It arises when there are imbalances in the execution of the operators running in parallel, resulting in some of the operators running for a longer time than others. The differences in execution times may cause some of the processors to remain idle while others still compute a part of the query. Execution skew can be a consequence of other forms of skew within a query, e.g., data skew, or arise because of temporally unavailable resources that affect the execution speed of a unit of work. The database system can minimize execution skew through the careful allocation of processors to operators and the selection of the appropriate parallel query plan. Execution skew arises in both operator-level parallelism as well as intra-operator parallelism. The interested reader is referred to [1] which presents a classification of skew effects in parallel database systems.

Cross-references

- ▶ Data Skew
- ▶ Intra-Operator Parallelism
- ▶ Operator-Level Parallelism

Recommended Reading

1. Märtens H. A Classification of Skew Effects in Parallel Database Systems. In Proc. 7th Int. Euro-Par Conference, 2001, pp. 291–300.

Exhaustivity

- ▶ Relevance

Existence Time

► Lifespan

Explicit Event

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition

In active databases, an explicit event is explicitly specified in the ECA rule definition.

Key Points

ECA rules were developed as an optimization of condition action rules. The performance of rule evaluation was improved by allowing, or even requiring, explicit definition of when rules should be triggered in the form of events. For example, (**ON** $update_A$ **followed_by** $update_B$ **IF** $A = 5 \wedge B = 3$ **DO** action) is a rule specification that is triggered by the explicit event $update_A$ **followed_by** $update_B$ (i.e., a sequence of updates).

Cross-references

► Atomic Event
► Composite Event
► ECA Rules
► Event
► Event Detection
► Event Specification
► Implicit Event

Exploratory Data Analysis

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Definition

Exploratory Data Analysis (EDA) is an *approach* to data analysis that employs a number of different techniques to:

1. Look at data to see what it seems to say,
2. Uncover underlying structures,
3. Isolate important variables,
4. Detect outliers and other anomalies,
5. Suggest suitable models for conventional statistics.

Key Points

The term "Exploratory Data Analysis" was introduced by John W. Tukey who in [2] shows how simple graphical and quantitative techniques can be used to open-mindedly explore data.

Typical graphical techniques are

1. Plotting the raw data (e.g., stem-and-leaf diagrams, histograms, scatter plots)
2. Plotting simple statistics (e.g., mean plots, box plots, residual plots)
3. Positioning (multiple) plots to amplify cognition

Typical quantitative techniques are

1. Interval estimation
2. Measures of location or of scale
3. Shapes of distributions

Exploratory data analysis can help to improve the results of statistical hypothesis testing by forcing one to look at data unbiased before formulating hypotheses which are subsequently tested using conventional statistics (confirmatory data analysis). Its techniques allow analysts to better assess assumptions on which statistical inference will be based and support the selection of appropriate statistical tools. EDA can also provide a background for further data collection.

For users of database systems it might be interesting to note that many EDA techniques have been adopted to support the exploration stage of data mining [1]. In particular, they are being used to identify the most relevant variables and to determine the complexity and general nature of models that can be taken into account for the model building and validation stages. Data mining and EDA are complementary in the sense that the former is more oriented towards applications and the later can help understand the basic nature of the underlying phenomena.

Cross-references

► Data Visualization

Recommended Reading

1. Berry M.J.A. and Linoff G.S. Mastering Data Mining. Wiley, New York, 2000.
2. Tukey J.W. Exploratory Data Analysis. Addison Wesley, Reading, MA, 1977.

Exploring

► Browsing in Digital Libraries

Expressive Power of Query Languages

Leonid Libkin
University of Edinburgh, Edinburgh, UK

Definition

The study of expressive power concentrates on comparing classes of queries that can be expressed in different languages, and on proving expressibility – or *in*expressibility – of certain queries in a query language.

Historical Background

Ever since Codd proposed relational calculus (first-order predicate logic) as a basic relational query language, it has been common for database query languages to have limited expressiveness. If a language cannot express everything computable, then it is natural to ask:

1. What queries cannot be expressed in a language $\mathcal{L}$?
2. Which methods are available for proving such results?

Furthermore, if there are two query languages $\mathcal{L}_1$ and $\mathcal{L}_2$, one may want to compare their expressiveness: for example, $\mathcal{L}_1 \subsetneq \mathcal{L}_2$ means that all queries expressible in $\mathcal{L}_1$ are also expressible in $\mathcal{L}_2$, but there are queries expressible in $\mathcal{L}_2$ that are not expressible in $\mathcal{L}_1$.

In 1975, Fagin [4] showed that queries such as the transitive closure of a graph and connectivity test cannot be expressed in relational calculus. The 0-1 law for first-order logic [5] implies that relational calculus furthermore lacks the ability to count, which was confirmed by the design of SQL, that explicitly includes counting primitives (aggregate functions). The field of *finite model theory* [1,9,12] provided many tools for studying the expressiveness of query languages: for example, Ehrenfeucht-Fraïssé games of different kinds, locality, and automata-based techniques for more expressive languages over special classes of structures such as trees.

In the direction of comparing power of languages, one of the most popular lines of work is comparing the power of a language with the class of all queries that have certain data complexity [3]. Classical results along these lines include extensions of relational calculus that describe familiar complexity classes such as PTIME.

Foundations

Relational calculus and algebra have precisely the power of first-order logic (FO) over *finite* relational structures. Many classical tools from logic, however, are inapplicable to finite structures. Investigation of the power of FO that applies to both finite and infinite structures was initiated by Ehrenfeucht and Fraïssé who developed the technique of Ehrenfeucht-Fraïssé games for characterizing the expressiveness of FO. With this technique it is very easy to prove, for example, that the *parity* query (is the number of elements of a set even?) is not expressible in FO. A more complicated application of Ehrenfeucht-Fraïssé games shows that the parity query is not FO-definable even over linear orders.

While parity is a nice and simple example of a query that involves counting, early questions about expressiveness of relational query languages concentrated on queries such as the transitive closure of a graph. For example, if a database represents a management hierarchy, it is natural to ask whether employee x reports to employee y. Such a query, in the absence of a priori bounds on the possible length of a management chain, involves transitive closure computation.

In 1975, Fagin [4] showed that graph connectivity is not definable in FO. In fact he proved a stronger result that it is not even definable in existential monadic second-order logic, whose formulae are of the form $\exists X_1 \ldots \exists X_k\ \varphi$, where the X_i's range over *sets of nodes* and φ is an FO formula. Inexpressibility of graph connectivity implies inexpressibility of transitive closure: had transitive closure E^* of a graph E been expressible, so would be connectivity by $\forall x \forall y E^*(x, y)$.

Early results have been proved using Ehrenfeucht-Fraïssé game techniques. Later easier techniques have been developed, such as *locality* and *zero-one laws*. Locality says that a formula $\varphi(x_1,\ldots,x_n)$ cannot "see" far beyond its free variables. For example, the transitive closure is not local because in a graph with directed edges $(0, 1),(1, 2),(2, 3),\ldots,(n-1, n)$ a given formula $\varphi(x, y)$ will not see the difference between pairs (j, k) and (k, j) for $j < k$ if the numbers k and j as well as $k - j$ and $n - k$ are sufficiently large.

Zero-one laws say that asymptotically the truth value of a sentence is almost surely false or almost surely true. For example, *parity* violates this property, as the proportion of structures of cardinality n for which the *parity* query is true oscillates between 0 and 1. For surveys of these results, see [1,12].

Since FO cannot do nontrivial counting and cannot express fixed-point computation, database theory research looked in detail into such extensions of FO. The simplest way to add counting is by means of counting quantifiers $\exists i x \varphi(x)$ stating that there are at least i witnesses x of $\varphi(x)$. For example, one can express that the number of witnesses of $\varphi(x)$ is even as $\exists i \exists! i x \varphi(x) \wedge \exists j\ (j + j = i)$, where $\exists! i x\ \varphi(x)$ says that the number of witnesses is exactly i (it is defined as $\exists i x \varphi(x) \wedge \exists j (\exists j x \varphi(x) \wedge j > i)$).

One can then prove that such an extension of FO is still local, which means, in particular, that queries such as transitive closure and graph connectivity are not expressible. However, this form of counting is very different from the usual one in SQL, such as aggregate functions together with `GROUPBY` and `HAVING` clauses. These can be modeled by adding more powerful arithmetic and aggregate terms to FO. The idea is that if there are some aggregate functions $\mathcal{F}_1,\ldots,\mathcal{F}_k$ (such as `MIN`, `SUM`, `AVG`), one can have expressions of the form

$$\mathrm{AGGR}_l[i_1 : \mathcal{F}_1,\ldots,i_k : \mathcal{F}_k](e)$$

whose semantics is basically that of the following SQL query:

```
SELECT  #1,..., #m, F1(#i1),..., Fk(#ik)
FROM    E
GROUPBY #1,..., #l
```

where `E` is the result of the expression e, which is assumed to produce a relation with m attributes [10,11]. In fact even more general aggregate terms could be permitted, and even with them, the resulting queries that are expressed in such an extension remain local. Hence, even aggregate extensions of FO cannot express fixed-point computations required to compute queries such as the transitive closure [11].

However, for this result it is essential that no operations are permitted on the domain of graph nodes; for example, one cannot compare them and say $a < b$. If an ordering is available, then even with very basic aggregate functions, proving inexpressibility of transitive closure is at least as hard as solving some long-standing open problems related to separation of complexity classes.

Another well-studied extension of FO is that with fixed-point operators. For example, the following Datalog program computes the transitive closure of a graph represented by the edge relation e:

$$\begin{aligned} trcl(x, y) &:- e(x, y) \\ trcl(x, y) &:- e(x, z), \wedge trcl(z, y) \end{aligned}$$

This could also be expressed as a *least fixed-point* formula $\mathrm{LFP}_{R,x,y}\ (E(x, y) \vee \exists z(E(x, z) \wedge R(z, y)))$, which computes the least fixed-point of the operator that sends each binary relation R to the relation $F(R) = E \cup E \circ R$, where $\circ$ is the relational composition.

For such formulae, it is essential that R occur positively (under an even number of negations), but many different flavors of datalog and fixed-point logics are known, which permit more flexible syntax and fewer restrictions (often at the expense of a more complicated semantics).

It was proved that such an extension of FO with least fixed-point still has the zero-one law [2]. In particular, the *parity* query cannot be defined in it. In fact this continues to hold for more powerful fixed-point operators.

But similarly to the case of aggregates, the situation changes dramatically if a linear ordering is allowed on the domain. In that case, the Immerman-Vardi theorem states that the least-fixed-point extension of FO captures precisely the class of queries with PTIME data complexity [8,14]. And various other fixed-point logics have been proposed to capture complexity classes such as DLOGSPACE, NLOGSPACE, PSPACE over ordered structures [9].

This gives rise to a natural question whether it is possible to find a logical query language that captures polynomial-time queries over unordered structures. This question, asked for the first time in [3], remains open, despite multiple attempts to solve it. For example, adding both fixed-point operators and counting still falls short of polynomial time. See Chapts. 1 and 2 in [6] for surveys. There are some positive results when the class of structures is restricted: for example, a different (inflationary) fixed-point operator together with counting captures polynomial time on the class of all unordered planar graphs [7].

Questions related to the expressive power of query languages have been addressed for other data models as

well. Results on aggregates are closely related to the expressiveness of relational languages under the bag (multiset) semantics. Many results about expressiveness of languages for nested relations can be derived from the *conservativity* theorem of [15] stating a query from relations to relations, even if it is expressed with the help of the operators of *nested* relational algebra, can be expressed in FO without any such operators. Similar in flavor conservativity results have been proved for constraint databases, and extensions of relational calculus with various constraints, such as polynomial (in) equalities over the reals (see Chapt. 5 of [6]).

Questions related to the expressive power of query languages are also actively studied in the context of XML and query languages for unranked trees (see, e.g., [13] for an overview).

Key Applications

The inability of relational calculus (and more generally its extension with aggregation) to express fixed-point queries was behind the addition of recursive constructs to SQL3. For example, to find all descendants of a node *a* in a graph represented by a relation `Graph(source, destination)`, one can write in SQL3:

```
WITH RECURSIVE TrCl(ancestor,
descendant) AS
   ( (SELECT source as ancestor, desti
      nation as descendant FROM Graph)
  UNION
    (SELECT Graph.source as ancestor,
    TrCl.descendant
    FROM Graph, TrCl
    WHERE Graph.destination = TrCl.
    ancestor)
  )
SELECT descendant FROM TrCl where
ancestor='a'
```

Often it is the understanding of the expressiveness of a language that guides the design of additional features.

Cross-references

- ▶ Aggregation
- ▶ Data Complexity
- ▶ Datalog
- ▶ Ehrenfeucht-Fraïssé Games
- ▶ Locality
- ▶ Zero-One Laws

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
2. Blass A., Gurevich Y., and Kozen D. A zero-one law for logic with a fixed-point operator. Inform. Control, 67:70–90, 1985.
3. Chandra A. and Harel D. Structure and complexity of relational queries. J. Comput. Syst. Sci., 25:99–128, 1982.
4. Fagin R. Monadic generalized spectra. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 21:89–96, 1975.
5. Fagin R. Probabilities on finite models. J. Symbolic Logic, 41:50–58, 1976.
6. Grädel E., Kolaitis Ph., Libkin L., Marx M., Spencer J., Vardi M.Y., Venema Y., and Weinstein S. Finite Model Theory and its Applications. Springer, Berlin, 2007.
7. Grohe M. Fixed-point logics on planar graphs. In Proc. IEEE Symposium on Logic in Computer Science, 1998, pp. 6–15.
8. Immerman N. Relational queries computable in polynomial time (extended abstract). In Proc. ACM Symposium on Theory of Computing, 1982, pp. 147–152.
9. Immerman N. Descriptive Complexity. Springer, Berlin, 1998.
10. Klug A. Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM, 29:699–717, 1982.
11. Libkin L. Expressive power of SQL. Theor. Comput. Sci., 296:379–404, 2003.
12. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.
13. Libkin L. Logics for unranked trees: an overview. Logic. Methods Comput. Sci., 2(3), 2006.
14. Vardi M.Y. The complexity of relational query languages. In Proc. ACM Symposium on Theory of Computing, 1982, pp. 137–146.
15. Wong L. Normal forms and conservative extension properties for query languages over collection types. J. Comput. Syst. Sci., 52:495–505, 1996.

Extended Entity-Relationship Model

Bernhard Thalheim
Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

EERM, HERM; Higher-Order Entity-Relationship Model; Hierarchical Entity-Relationship Model

Definition

The extended entity-relationship (EER) model is a language for defining the structure (and functionality) of database or information systems. Its structure is developed inductively. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set

constructors to attributes that have already been constructed. Entity types conceptualize structuring of things of reality through attributes. Cluster types generalize types or combine types into singleton types. Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined for a type. Typical integrity constraints of the extended entity-relationship model are participation, look-across, and general cardinality constraints. Entity, cluster, and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

Historical Background

The entity-relationship (ER) model was introduced by P.P. Chen in 1976 [1]. The model conceptualizes and graphically represents the structure of the relational model. It is currently used as the main conceptual model for database and information system development. Due to its extensive usage a large number of extensions to this model were proposed in the 80s and 90s. Cardinality constraints [1,3,4,8] are the most important generalization of relational database constraints [7]. These proposals have been evaluated, integrated or explicitly discarded in an intensive research discussion. The semantic foundations proposed in [2,5,8] and the various generalizations and extensions of the entity-relationship model have led to the introduction of the higher-order or hierarchical entity-relationship model [8] that integrates most of the extensions and also supports conceptualization of functionality, distribution [9], and interactivity [6] for information systems. Class diagrams of the UML standard are a special variant of extended entity-relationship models.

The ER conferences (annually; since 1996: International Conference on Conceptual Modeling, http://www.conceptualmodeling.org/) are the main fora for conceptual models and modeling.

Foundations

The extended entity-relationship model is mainly used as a language for conceptualization of the structure of information systems applications. Conceptualization of database or information systems aims to represent the logical and physical structure of an information system. It should contain all the information required by the user and required for the efficient behavior of the whole information system for all users. Conceptualization may further target the specification of database application processes and the user interaction. Structure descriptions are currently the main use of the extended ER model.

An Example of an EER Diagram

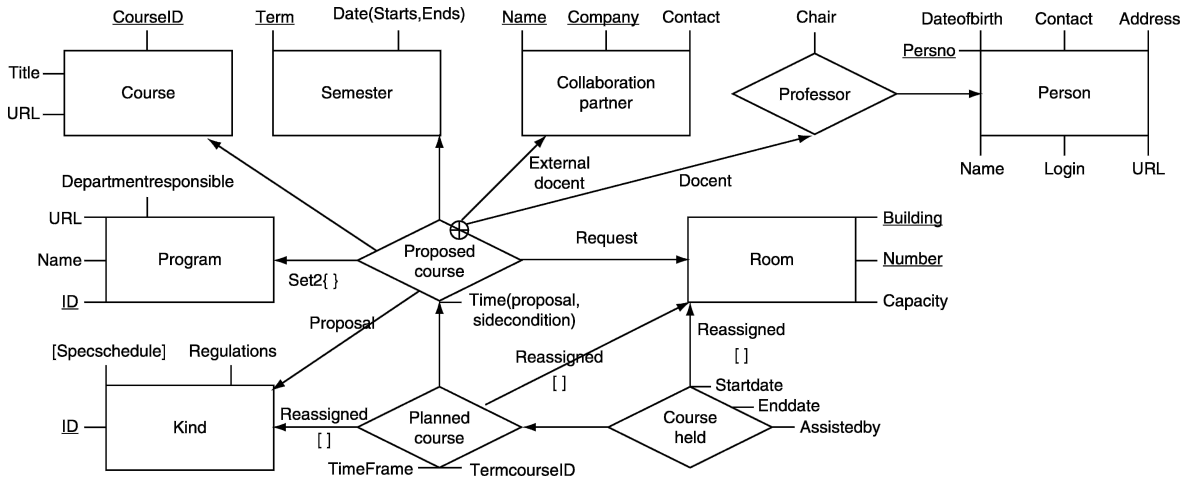
The EER model uses a formal language for schema definition and diagrams for graphical representation of the schema. Let us consider a small university application for management of *Courses*. *Proposed courses* are based on courses and taught by a docent or an external docent within a certain semester and for a set of programs. Proposals typically include a request for a room and for a time and a categorization of the kind of the course. These proposals are the basis for course planning. Planning may change time, room and kind. Planned courses are held at the university. Rooms may be changed. The example is represented by the EER diagram in Fig. 1.

Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding entity or relationship type. Attributes primarily identifying a type are underlined. Relationship types are represented graphically by diamonds and associated by directed arcs to their components. A cluster type is represented by a diamond, is labeled by the disjoint union sign, and has directed arcs from the diamond to its component types. Alternatively, the disjoint union representation $\oplus$ is attached to the relationship type that uses the cluster type. In this case directed arcs associate the $\oplus$ sign with component types. An arc may be annotated with a label.

The Definition Scheme for Structures

The extended entity-relationship model uses a data type system for its attribute types. It allows the construction of entity types $E \triangleq (attr(E), \Sigma_E)$ where E is the entity type defined as a pair – the set $attr(E)$ of attribute types and the set Σ_E of integrity constraints that apply to E. The definition def of a type T is denoted by $T \triangleq def$.

The EER model lets users inductively build relationship types $R \triangleq (T_1,...,T_n, attr(R), \Sigma_R)$ of order i $(i \geq 1)$ through a set of (labeled) types of order less than i, a set of attribute types, and a set of integrity constraints that apply to R. The types $T_1,...,T_n$ are the components of the relationship type. Entity types are of order 0.



Extended Entity-Relationship Model. Figure 1. Extended Entity-Relationship Diagram for Course Management.

Relationship types are of order 1 if they have only entity types as component types. Relationship types are of order i if all component types are of order less than i and if one of the component types is of order $i-1$.

Additionally, cluster types $C \stackrel{\circ}{=} T_1 \dot{\cup} ... \dot{\cup} T_n$ of order i can be defined through a disjoint union $\dot{\cup}$ of relationship types of order less than i or of entity types.

Entity/relationship/cluster classes T^C contain a set of objects of the entity/relationship/cluster type T. The EER model mainly uses set semantics, but (multi-)list or multiset semantics can also be used. Integrity constraints apply to their type and restrict the classes. Only those classes are considered for which the constraints of their types are valid. The notions of a class and of a type are distinguished. Types describe the structure and constraints. Classes contain objects.

The data type system is typically inductively constructed on a base type B by application of constructors such as the tuple or products constructor (..), set constructor {..}, and the list constructor $< .. >$. Types may be optional component types and are denoted by [..].

The types T can be labeled $l : T$. The label is used as an alias name for the type. Labels denote roles of the type. Labels must be used if the same type is used several times as a component type in the definition of a relationship or cluster type. In this case they must be unique.

An entity-relationship schema consists of a set of data, attribute, entity, relationship, and cluster types which types are inductively built on the basis of the base types.

Given a base type system B. The types of the ER schema are defined through the type equation: $T = B|(l_1 : T, ..., l_n : T)\ |\{T\}\ |\ < \mathrm{T} >\ |\ [\mathrm{T}]\ |\mathrm{T} \dot{\cup} \mathrm{T}|\mathrm{l} : T|N \stackrel{\circ}{=} T$

Structures in Detail

The classical four-layered approach is used for inductive specification of database structures. The first layer is the data environment, called the basic data type scheme, which is defined by the system or is the assumed set of available basic data types. The second layer is the schema of a database. The third layer is the database itself representing a state of the application's data often called micro-data. The fourth layer consists of the macro-data that are generated from the micro-data by application of view queries to the micro-data.

Attribute Types and Attribute Values The classical ER model uses basic (first normal form) attributes. Complex attributes are inductively constructed by application of type constructors such as the tuple constructor (..), set constructor {..}, and the list constructor $< .. >$. Typical base types are integers, real numbers, strings, and time. Given a set of names $\mathcal{N}$ and a set of base types B, a basic attribute type $A :: B$ is given by an (attribute) name $A \in \mathcal{N}$ and a base type B. The association between the attribute name and the underlying type is denoted by ::. The base type B is often called the domain of A, i.e., $dom(A) = B$. Complex attributes are constructed on base attributes by application of the type constructors. The notion of a domain is extended to complex attributes, i.e., the domain of the complex attribute A is given by $dom(A)$. Components of complex attributes may be optional, e.g., the *Title* in the attribute *Name*.

Typical examples of complex and basic attributes in Fig. 1 are

Name ≙ *(FirstNames <FirstName>, FamName, [AcadTitles], [FamilyTitle])*,
PersNo ≙ *EmplNo* ∪̇ *SocSecNo*,
AcadTitles ≙ *{AcadTitle}*,
Contact ≙ *(Phone({PhoneAtWork}, private), Email, URL, WebContact, [Fax({PhoneAtWork})])*,
PostalAddress ≙ *(Zip, City, Street, HouseNumber)* for *DateOfBirth :: date, AcadTitle :: acadTitleType, FamilyTitle :: familyTitleAcronym, Zip :: string7, SocSecNo :: string9, EmplNo :: int, City :: varString, Street :: varString, HouseNumber :: smallInt.*

The complex attribute *Name* is structured into a sequence of first names, a family name, an optional complex set-valued attribute for academic titles, and an optional basic attribute for family titles. Academic titles and family titles can be distinguished from each other.

Entity Types and Entity Classes Entity types are characterized by their attributes and their integrity constraints. Entity types have a subset K of the set of attributes which serve to identify the objects of the class of the type. This concept is similar to the concept of key known for relational databases. The key is denoted by ID(K). The set of integrity constraints Σ_E consists of the keys and other integrity constraints. Identifying attributes may be underlined instead of having explicit specification.

Formally, an entity type is given by a name E, a set of attributes $attr(E)$, a subset $id(E)$ of $attr(E)$, and a set Σ_E of integrity constraints, i.e.,

$E \triangleq (attr(E), \Sigma_E)$.

The following types are examples of entity types in Fig. 1:

Person ≙ *({Name, Login, URL, Address, Contact, DateOfBirth,* PersNo*})*
Course ≙ *({CourseID, Title, URL}, {ID({CourseID})})*,
Room ≙ *({Building, Number, Capacity}, {ID({Building, Number})})*,
Semester ≙ *({Term, Date(Starts, Ends)}, { ID({ Term })})*.

An ER schema may use the same attribute name with different entity types. For instance, the attribute *URL* in Fig. 1 is used for characterizing additional information for the type *Person* and the type *Course*. If they need to be distinguished, then complex names such as *CourseURL* and *PersonURL* are used.

Objects on type E are tuples with the components specified by a type. For instance, the object (or *entity*) *(HRS3, 408A, 15)* represents data for the *Room* entity type in Fig. 1.

An entity class E^C of type E consists of a finite *set* of objects on type E for which the set Σ_E of integrity constraints is valid.

Cluster Types and Cluster Classes A disjoint union ∪̇ of types whose identification type is domain compatible is called a cluster. Types are domain compatible if they are subtypes of a common more general type. The union operation is restricted to disjoint unions since identification must be preserved. Otherwise, objects in a cluster class cannot be related to the component classes of the cluster type. Cluster types can be considered as a generalization of their component types.

A cluster type (or "category") $C \stackrel{\circ}{=} l_1 : R_1 \dot{\cup} l_2 : R_2 \dot{\cup} \ldots \dot{\cup} l_k : R_k$ is the (labeled) disjoint union of types $R_1, \ldots, R_k$. Labels can be omitted if the types can be distinguished.

The following type is an example of a cluster type:

Teacher ≙ *ExternalDocent : CollaborationPartner* ∪̇ *Docent : Professor.*

The cluster class C^C is the 'disjoint' union of the sets $R_1^C, \ldots, R_k^C$. It is defined if ${R_1}^C, \ldots {R_k}^C$ are disjoint on their identification components. If the sets $R_1^C, \ldots, R_k^C$ are not disjoint then labels are used for differentiating the objects of clusters. In this case, an object uses a pair representation (l_i, o_i) for objects o_i from ${R_i}^C$.

Relationship Types and Relationship Classes First order relationship types are defined as associations between entity types or clusters of entity types. Relationship types can also be defined on the basis of relationship types that are already defined. This construction must be inductive and cannot be cyclic. Therefore, an order is introduced for relationship types. Types can only be defined on the basis of types which have a lower order. For instance, the type *Professor* in Fig. 1 is of order 1. The type *ProposedCourse* is of order 2 since all its component types are either entity types or types of order 1. A relationship type of order i is defined as an association of relationship types of order less than i or of entity types. It is additionally required that at least one of the component types is of

order $i-1$ if $i>1$. Relationship types can also be characterized by attributes. Relationship types with one component type express a subtype or an Is-A relationship type. For instance, the type *Professor* is a subtype of the type *Person*.

Component types of a relationship type may be labeled. Label names typically provide an understanding of the role of a component type in the relationship type. Labeling uses the definition scheme *Label* : *Type*. For instance, the *Kind* entity type is labeled by *Proposal* for the relationship type *ProposedCourse* in Fig. 1.

Cluster types have the maximal order of their component types. Relationship types also may have cluster type components. The order of cluster type components of a relationship type of order i must be less than i.

Component types that are not used for identification within the relationship type can be optional. For instance, the *Room* component in Fig. 1 is optional for the type *PlannedCourse*. If the relationship object in the *PlannedCourse* class does not have a room then the proposal for rooms in *ProposedCourse* is accepted. A specific extension for translation of optional components may be used. For instance, *Room* in Fig. 1 is inherited to *PlannedCourse* from *ProposedCourse* if the *Room* component for a *PlannedCourse* is missing.

Higher order types allow a convenient description of types that are based on other types. For example, consider the course planning application in Fig. 1. Lectures are courses given by a professor or a collaboration partner within a semester for a number of programs. Proposed courses extend lectures by describing which room is requested and which time proposals and which restrictions are made. Planing of courses assigns a room to a course that has been proposed and assigns a time frame for scheduling. The kind of the course may be changed. Courses that are held are based on courses planned. The room may be changed for a course. The following types specify these assertions.

ProposedCourse $\triangleq$ (*Teacher*, *Course*, *Proposal : Kind*, *Request : Room*, *Semester*, *Set2 : {Program}*, *{Time(Proposal, SideCondition)}*, $\Sigma_{ProposedCourse}$),

PlannedCourse $\triangleq$ (*ProposedCourse*, *[Reassigned : Kind]*, *[Reassigned : Room]*, *{TimeFrame, TermCourseID}*, $\Sigma_{PlannedCourse}$),

CourseHeld $\triangleq$ (*PlannedCourse*, *[Reassigned : Room]*, *{StartDate, EndDate, AssistedBy}*, $\Sigma_{CourseHeld}$).

The second and third types use optional components in case a proposal or a planning of rooms or kinds is changed. Typically, planned courses are identified by their own term-specific identification. Integrity constraints can be omitted until they have been defined.

Formally, a relationship type is given by a name R, a set $compon(R)$ of labeled components, a set of attributes $attr(R)$, and a set Σ_R of integrity constraints that includes the identification of the relationship type by a subset $id(R)$ of $compon(R) \cup attr(R)$, i.e., $R \triangleq (compon(R), attr(R), \Sigma_R)$.

It is often assumed that the identification of relationship types is defined exclusively through their component types. Relationship types that have only one component type are unary types. These relationship types define subtypes. If subtypes need to be explicitly represented then binary relationship types named by *IsA* between the subtype and the supertype are used. For instance, the type *Professor* in Fig. 1 is a subtype of the type *Person*.

An object (or a "relationship") on the relationship type $R \triangleq (R_1,...,R_n, \{B_1,...,B_k\}, id(R), \Sigma_R)$ is an element of the Cartesian product $R_1^C \times ... \times R_n^C \times dom(B_1) \times ... \times dom(B_k)$. A relationship class R^C consists of a finite *set* $R^C \subseteq R_1^C \times ... \times R_n^C \times dom(B_1) \times ... \times dom(B_k)$ of objects on R for which $id(R)$ is a key of R^C and which obeys the constraints Σ_R.

Integrity Constraints Each database model also uses a set of implicit model-inherent integrity constraints. For instance, relationship types are defined over their component types, and a (relationship) object presumes the existence of corresponding component objects. Typically only finite classes are considered. The EER schema is acyclic. Often names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal semantics allows us to derive synonym, homonym, antonym, toponym, hypernym, and holynym associations among the constructs used.

The most important class of integrity constraints of the EER model is the class of cardinality constraints. Other classes of importance for the EER model are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[5]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints.

Three main kinds of cardinality constraints are distinguished: participation constraints, look-across constraints, and general cardinality constraints. Given a relationship type $R \triangleq (compon(R), attr(R), \Sigma_R)$, a component R' of R, the remaining substructure $R'' = R \backslash R'$ and the remaining substructure $R''' = R'' \sqcap_R compon(R)$ without attributes of R.

The participation constraint $card(R, R') = (m, n)$ restricts the number of occurrences of R' objects in the relationship class R^C by the lower bound m and the upper bound n. It holds in a relationship class R^C if for any object $o' \in R'^C$ there are at least m and at most n objects $o \in R^C$ with $\pi_{R'}(o) = o'$ for the projection function $\pi_{R'}$ that projects o to its R' components.

Participation constraints relate objects of relationship classes to objects of their component classes. For instance, the constraint *card*(*ProposedCourse*, *SemesterCourse*) = (0, 3) restricts relationship classes for proposals for courses per semester to at least 0 and at most 3, i.e., each course is proposed at most three times in a semester. There are at most three objects o in $ProposedCourse^C$ with the same course and semester objects. The integrity constraint *card*(*ProposedCourse*, *DocentSemester*) = (3, 7) requires that each docent is giving at least 3 courses and at most 7 courses. External docents may be obliged by other restrictions, e.g., *card*(*ProposedCourse*, *ExternalDocentSemester*) = (0,1).

Formally, the integrity constraint $card(R,R') = (m,n)$ is valid in R^C if $m \leq |\{o \in R^C : \pi_{R'}(o) = o'\}| \leq n$ for any $o' \in \pi_{R'}(R^C)$ and the projection $\pi_{R'}(R^C)$ of R^C to R'.

If $card(R, R') = (0, 1)$ then R' forms an identification or a key of R, i.e., $ID(R')$ for R. This identification can also be expressed by a functional dependency $R : R' \rightarrow R''$.

The lookup or look-across constraint $look(R, R') = m..n$ describes how many objects o''' from R'''^C may potentially 'see' an object o' from R'^C. It holds in a relationship class R^C if for any object $o''' \in dom(R''')$ there are at least m and at most n related objects o' with $\pi_{R'}(o) = o'$, i.e., $m \leq |\{o' \in \pi_{R'}(R^C) : o \in R^C \wedge \pi_{R'}(o) = o' \wedge \pi_{R'''}(o) = o'''\}| \leq n$ for any $o''' \in Dom(R''')$. Typically, look-across constraints are used for components consisting of one type. Look-across constraints are not defined for relationship types with one component type.

Look-across constraints are less intuitive for relationship types with more than two component types or with attribute types. For instance, the look-across constraint *look*(*ProposedCourse*, *DocentSemester*) = 0..7 specifies that for any combination of *Teacher*, *Room*, *Kind*, and *Program* objects there are between 0 and 7 *Docent* and *Semester* combinations. The lower bound expresses that there are *Teacher*, *Room*, *Kind*, and *Program* which do not have a *Docent* and *Semester* combination.

Look-across constraints for a binary relationship type whose component types form a key of the relationship type can equivalently be expressed by participation constraints, i.e., $look(R, R_1) = m_1..n_1$ if and only if $card(R, R_2) = (m_1, n_1)$. Similarly, $look(R, R_2) = m_2..n_2$ if and only if $card(R, R_1) = (m_2, n_2)$. This equivalence is neither valid for binary relationship types which cannot by identified by their components nor for relationship types with more than two components.

Participation and look-across constraints can be extended to substructures and intervals and to other types such as entity and cluster types. Given a relationship type R, a substructure R' of R, R'' and R''' as above, and given furthermore an interval $I \subseteq \mathbb{N}_0$ of natural numbers including 0, the (general) cardinality constraint $card(R, R') = I$ holds in a relationship class R^C if for any object $o' \in \pi_{R'}(R^C)$ there are $i \in I$ objects o with $\pi_{R'}(o) = o'$, i.e., $|\{o \in R^C : \pi_{R'}(o) = o'\}| \in I$ for any $o' \in \pi_{R'}(R^C)$.

The following participation, look-across and general cardinality constraints are examples in Fig. 1:

For any $R' \in$ {Semester, Course, Kind} *card*(*ProposedCourse*,R') = (0, n),
card(*ProposedCourse*, *SemesterCourseTeacher*) = (0, 1),
card(*CourseHeld*, *PlannedCourse*) = (1, 1),
card(*PlannedCourse*, *ProposedCourse*[*Semester*]*Room TimeFrame*) = (0, 1),
card(*ProposedCourse*, *DocentSemester*) = {0, 3, 4, 5, 6, 7}.

The first constraint does not restrict the database. The second constraint expresses a key or functional dependency. The types *Semester Course Teacher* identify any of the other types in the type *ProposedCourse*, i.e.,

ProposedCourse: *{Semester, Course, Teacher}* → *{Request, Time, Proposal, Set2}*.

The third constraint requires that any planned course must be given. The fourth constraint requires that rooms are not overbooked. The fifth constraint allows that docents may not teach in a semester, i.e., have a sabbatical. If a docent is teaching

in a semester then at least 3 and at most 7 courses are given by the docent.

Look-across constraints were originally introduced by Chen [1] as cardinality constraints. UML uses look-across constraints. Participation and look-across constraints cannot be axiomatized through a Hilbert- or Gentzen-type logical calculus. If only upper bounds are of interest then an axiomatization can be found in [3] and [4]. General cardinality constraints combine equality-generating and object-generating constraints such as keys, functional dependencies and referential integrity constraints into a singleton construct.

Logical operators can be defined for each type. A set of logical formulas using these operators can define the integrity constraints which are valid for each object of the type.

Schemata

The schema is based on a set of base (data) types which are used as value types for attribute types.

A set $\{E_1,...E_n, C_1,...,C_l, R_1,...,R_m\}$ of entity, cluster and (higher-order) relationship types on a data scheme DD is called schema if the relationship and cluster types use only the types from $\{E_1,..., E_n, C_1,...,C_l, R_1,...,R_m\}$ as components and cluster and relationship types are properly layered.

An EER schema is defined by the pair $\mathcal{D} = (\mathcal{S}, \Sigma)$ where $\mathcal{S}$ is a schema and Σ is a set of constraints. A database $\mathcal{D}^C$ on $\mathcal{D}$ consists of classes for each type in $\mathcal{D}$ such that the constraints Σ are valid.

The classes of the extended ER model have been defined through sets of objects on the types. In addition to sets, lists, multi-sets or other collections of objects may be used. In this case, the definitions used above can easily be extended [8].

A number of domain-specific extensions have been introduced to the ER model. One of the most important is the extension of the base types by spatial data types such as: point, line, oriented line, surface, complex surface, oriented surface, line bunch, and surface bunch. These types are supported by a large variety of functions such as: meets, intersects, overlaps, contains, adjacent, planar operations, and a variety of equality predicates.

The translation of the schema to (object-)relational or XML schemata can be based on a profile [4]. Profiles define which translation choice is preferred over other choices, how hierarchies are treated, which redundancy and null-value support must be provided, which kind of constraint enforcement is preferred, which naming conventions are chosen, which alternative for representation of complex attributes is preferred for which types, and whether weak types can be used. The treatment of optional components is also specified through the translation profile of the types of the schema. A profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.

The EER schema can be used to define views. The generic functions insert, delete, update, projection, union, join, selection and renaming can be defined in a way similarly to the relational model. Additionally, nesting and unnesting functions are used. These functions form the algebra of functions of the schema and are the basis for defining queries. A singleton view is defined by a query that maps the EER schema to new types. Combined views also may be considered which consist of singleton views which together form another EER schema.

A view schema is specified over an EER schema $\mathcal{D}$ by a schema $\mathcal{V} = \{S_1,...,S_m\}$, an auxiliary schema $\mathcal{A}$ and a (complex) query $q : \mathcal{D} \times \mathcal{A} \rightarrow \mathcal{V}$ defined on $\mathcal{D}$ and $\mathcal{A}$. Given a database $\mathcal{D}^C$ and the auxiliary database $\mathcal{A}^C$. The view is defined by $q(\mathcal{D}^C \times \mathcal{A}^C)$.

Graphical Representation

The schema in Fig. 1 consists of entity, cluster and relationship types. The style of drawing diagrams is one of many variants that have been considered in the literature. The main difference of representation is the style of drawing unary types. Unary relationship types are often represented by rectangles with rounded corners or by (directed) binary IsA-relationship types which associate by arcs the supertype with the subtype. Tools often do not allow cluster types and relationship types of order higher than one. In this case, those types can be objectified, i.e., represented by a new (abstract) entity type that is associated through binary relationship types to the components of the original type. In this case, identification of objects of the new type is either inherited from the component types or is provided through a new (surrogate) attribute. The first option results in the introduction of so-called weak types. The direct translation of these weak types to object-relational models must be combined with the introduction of rather complex

constraint sets. Typically, this complexity can be avoided if the abstract entity type is mapped together with the new relationship types to a singleton object-relational type. This singleton type is also the result of a direct mapping of the original higher-order relationship type.

The diagram can be enhanced by an explicit representation of cardinality and other constraints. If participation constraints $card(R, R') = (m, n)$ are used for component consisting of one type R' then the arc from R to R' is labeled by (m, n). If look-across constraints $look(R, R') = m..n$ are used for binary relationship types then the arc from R to R' is labeled by $m..n$.

Key Applications

The main application area for extended ER models is the conceptualization of database applications.

Database schemata can be translated to relational, XML or other schemata based on transformation profiles that incorporate properties of the target systems.

Future Directions

The ER model has had a deep impact on the development of diagramming techniques in the past and is still influencing extensions of the unified modeling language UML. UML started with binary relationship types with look-across constraints and without relationship type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow for distinguishing generalization from specialization.

ER models are not supported by native database management systems and are mainly used for modeling of applications at the conceptual or requirements level. ER schemata are translated to logical models such as XML schemata or relational schemata or object-relational schemata. Some of the specifics of the target models are not well supported by ER models and must be added after translating ER schemata to target schemata, e.g., specific type semantics such as list semantics (XML) or as special ordering or aggregation treatment of online analytical processing (OLAP) applications.

The ER model has attracted a lot of research over the last 30 years. Due to novel applications and to evolution of technology old problems and novel problems are challenging the research on this model. Typical old problems that are still not solved in a satisfactory manner are: development of a science of modeling, quality of ER schemata, consistent refinement of schemata, complex constraints, normalization of ER schemata, normalization of schemata in the presence of incomplete constraint sets. Novel topics for ER research are for instance: evolving schema architectures, collaboration of databases based on collaboration schemata, layered information systems and their structuring, schemata with redundant types, ER schemata for OLAP applications.

Structures of database applications are often represented through ER models. Due to the complexity of applications, a large number of extensions have recently been proposed, e.g., temporal data types, spatial data types, OLAP types and stream types. Additionally, database applications must be integrated and cooperate in a consistent form. The harmonization of extensions and the integration of schemata is therefore a never ending task for database research.

ER models are currently extended for support of (web) content management that is based on structuring of data, on aggregation of data, on extending data by concepts and on annotating data sets for simple reference and usage. These applications require novel modeling facilities and separation of syntactic, semantic and pragmatic issues. The ER model can be extended to cope with these applications.

The ER model is mainly used for conceptual specification of database structuring. It can be enhanced by operations and a query algebra. Operations and the queries can also be displayed in a graphical form, e.g., on the basis of VisualSQL. Most tools supporting ER models do not currently use this option. Enhancement of ER models by functionality is necessary if the conceptualization is used for database development. Based on functionality enhancement, view management facilities can easily be incorporated into these tools.

ER models are becoming a basis for workflow systems data. The standards that have been developed for the specification of workflows have not yet been integrated into sophisticated data and application management tools.

URL to Code

http://www.informatik.uni-kiel.de/\~thalheim/HERM.htm

http://www.is.informatik.uni-kiel.de/~thalheim/indeeerm.htm

Readings on the RADD project (Rapid Application and Database Development) Authors: M. Albrecht, M. Altus, E. Buchholz, H. Cyriaks, A. Düsterhöft,

J. Lewerenz, H. Mehlan, M. Steeg, K.D. Schewe, and B. Thalheim.

Cross-references

- ▶ Entity Relationship Model
- ▶ Relational Model
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading

1. Chen P.P. The entity-relationship model: toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
2. Gogolla M. An Extended Entity-Relationship Model – Fundamentals and Pragmatics. LNCS 767. Springer, Berlin Heidelberg New York, 1994.
3. Hartmann S. Reasoning about participation constraints and Chen's constraints. In Proc. 14th Australasian Database Conf., 2003, pp. 105–113.
4. Hartmann S., Hoffmann A., Link S., and Schewe K.-D. Axiomatizing functional dependencies in the higher-order entity-relationship model. Inf. Process. Lett., 87(3):133–137, 2003.
5. Hohenstein U. Formale Semantik eines erweiterten Entity-Relationship-Modells. Teubner, Stuttgart, 1993.
6. Schewe K.-D. and Thalheim B. Conceptual modelling of web information systems. Data Knowl. Eng., 54:147–188, 2005.
7. Thalheim B. Dependencies in Relational Databases. Teubner, Leipzig, 1991.
8. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, Berlin Heidelberg New York, 2000.
9. Thalheim B. Codesign of structuring, functionality, distribution and interactivity. In Proc. 1st Asian-Pacific Conf. on Conceptual Modeling, 2004, pp. 3–12.

Extended Functional Dependencies

- ▶ Functional Dependencies for Semi-structured Data

Extended Relations

- ▶ Conditional Tables
- ▶ Naive Tables

Extended Transaction Models

- ▶ Generalization of ACID Properties
- ▶ Open Nested Transaction Models

Extended Transaction Models and the ACTA Framework

Panos K. Chrysanthis[1], Krithi Ramamritham[2]
[1]University of Pittsburgh, Pittsburgh, PA, USA
[2]Indian Institute of Technology Bombay, Mumbai, India

Synonyms

Advanced transaction models; Generalization of ACID properties

Definition

Although powerful, the transaction model adopted in traditional database systems is found lacking in functionality and performance when used for applications that involve reactive (endless), open-ended (long-lived) and collaborative (interactive) activities. Hence, various extensions to the traditional model have been proposed, referred to as *extended transactions*. These models are characterized by the structure of their transactions, the commit and abort dependencies and the visibility rules among transactions. *ACTA* is a comprehensive transaction framework that facilitate the specification, analysis and synthesis of extended transaction models. The name *ACTA*, meaning *actions* in Latin, was chosen given the framework's appropriateness for expressing the properties of actions used to compose a transactional computation.

Key Points

By means of the notion of transactions, database systems offer reliability guarantees concerning the correctness of data in spite of failures and concurrent accesses by multiple users. However, the transaction model as well as the simple data model adopted in traditional database systems have been found lacking in *functionality* and *performance* in their support of the emerging advanced database applications such as design databases, computer publishing, network management, multidatabases and mobile databases. In order to deal with the inherent limitations of the traditional data and atomic transaction model, researchers have proposed semantic and object-oriented data models and extensions to the traditional transaction model. Nested transactions was the first such extension that added a hierarchical structure to the traditional flat atomic

transactions. The hierarchical structure allows concurrency within a transaction and fine-grained failure and exception handling since subtransactions can abort independently without causing the abortion of the whole transaction.

The original nested transaction model was subsequently enhanced with new types of subtransactions, relaxed abort and commit dependencies and visibility rules for externalizing partial results among transactions. These extensions led to a variety of open-nested transactions models such as Sagas, Split Transactions, Flex Transactions, ConTracts and S-transactions, and of correctness criteria such as quasi serializability, epsilon-serializability, semantic atomicity, quasi failure-atomicity.

All the above extensions have been introduced with specific applications or with specific transaction properties in mind [2]. Their ad hoc character makes it difficult to identify the properties of transactions that adhere to a particular model and to ascertain in what respects an extended transaction model is similar or different from another. The need for a comprehensive transaction framework that would facilitate the precise specification of the properties of a model, vis a vis visibility, consistency, recovery and permanence, and allow the formal comparison of different models led to the development of *ACTA* [1]. ACTA is a first-order logic based formalism with a precedence relation that allows a transaction modeler to specify both the high level properties (requirements) of a model and the lower level behavioral aspects of the model in terms of axioms. Specifications include the following four components: (i) the set of transaction management events associated with the transaction model, such as *begin, commit, abort, split*, and *join*; (ii) the semantics of these significant events, characterized in terms of their effect on objects (their value and synchronization state) and other transactions (different types of dependencies, such as commit dependency and abort dependency); (iii) the view of each transaction, specifying the state of objects visible to that transaction; and (iv) the conflict set of each transaction, containing those operations with respect to which conflicts need to be considered.

Besides supporting the specification and analysis of existing transaction models, ACTA has the power to specify the requirements of new transactional applications and synthesize models that satisfy these requirements. This was demonstrated by deriving new transaction definitions either by starting from first principles or by modifying and/or combining the specifications of existing transaction models. The exercise of analyzing and synthesizing different transaction models revealed the many advantages of using a simple formalism like ACTA to deal with advanced transactions and has influenced a lot of transaction processing work in industry and academia.

Although ACTA has been developed to characterize extended transaction models, it has been extended to express the various correctness criteria beyond serializability. The use of this formalism resulted in a consolidated notion of correctness in which the different serializability-based criteria, such as predicatewise serializability and cooperative serializability, can been seen as special cases [3].

Cross-references

- ▶ ACID Properties
- ▶ Correctness Criteria beyond Serializability
- ▶ ConTract
- ▶ Compensating Transactions
- ▶ e-Commerce Transactions
- ▶ Flex Transactions
- ▶ Generalization of ACID Properties
- ▶ Internet Transactions
- ▶ Multilevel Transactions and Object-Model Transactions
- ▶ Nested Transaction Models
- ▶ Open Nested Transaction Models
- ▶ Polytransactions
- ▶ Sagas
- ▶ Semantic Atomicity
- ▶ Split Transactions
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transactional Processes
- ▶ Workflow Transactions

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19(3):450–491, 1994.
2. Elamagarmid A. K. (Ed.). Database Transaction Models for Advanced Applications, Morgan Kaufmann, Los Altos, CA, 1992.
3. Ramamritham K. and Chrysanthis P.K. A taxonomy of correctness criteria in database applications. VLDB J., 4(1):181–293, 1996.

Extendible Hashing

Donghui Zhang[1], Yannis Manolopoulos[2], Yannis Theodoridis[3], Vassilis J. Tsotras[4]
[1]Northeastern University, Boston, MA, USA
[2]Aristotle University, Thessaloniki, Greece
[3]University of Piraeus, Piraeus, Greece
[4]University of California – Riverside, Riverside, CA, USA

Definition

Extendible hashing is a dynamically updateable disk-based index structure which implements a hashing scheme utilizing a directory. The index is used to support exact match queries, i.e., find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), Extendible Hashing has better expected query cost O(1) I/O. Compared with linear hashing, extendible hashing does not have any overflow page. Overflows are handled by doubling the directory which logically doubles the number of buckets. Physically, only the overflown bucket is split.

Historical Background

The extendible hashing scheme was introduced by [1]. A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets and caches. Like arrays, hash tables have O(1) lookup cost on average.

Foundations

Structure

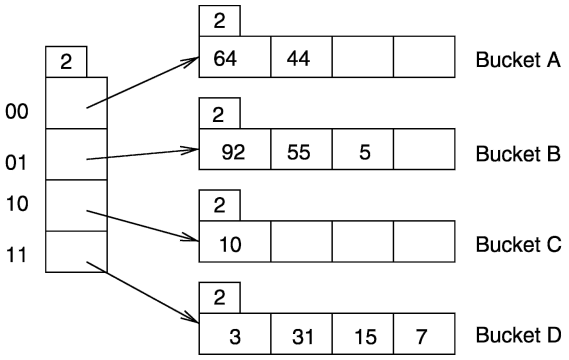
Extendible hashing uses a directory to access its buckets. This directory is usually small enough to be kept in main memory and has the form of an array with 2^d entries, each entry storing a bucket address (pointer to a bucket). The variable d is called the global depth of the directory. To decide where a key k is stored, extendible hashing uses the last d bits of some adopted hash function $h(k)$ to choose the directory entry. Multiple directory entries may point to the same bucket. Every bucket has a local depth *leqd*. The difference between local depth and global depth affects overflow handling.

An example of extendible hashing is shown in Fig. 1. Here there are four directory entries and four buckets. The global depth and all the four local depths are 2. For simplicity assume the adopted hash function is $h(k) = k$. For instance, to search for record 15, one refers to directory entry 15% 4 = 3 (or 11 in binary format), which points to bucket D.

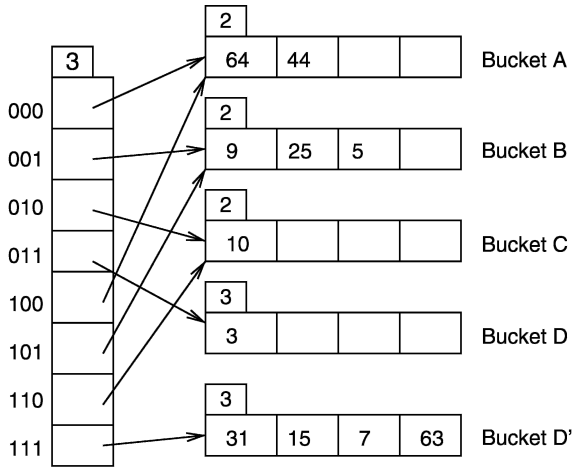
Overflow Handling

If a bucket overflow happens, the bucket is split into two. The directory may or may not double, depending on whether the local depth of the overflown bucket was equal to the global depth before split.

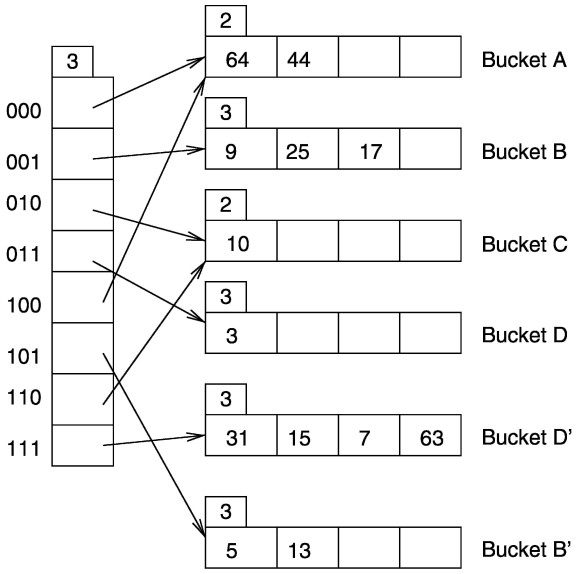
If the local depth was equal to global depth, d bits are not enough to distinguish the search values of the overflown bucket. Thus a directory *doubling* occurs, which effectively uses one more bit from the hash value. The directory size is then doubled (this does not mean that the number of buckets is doubled as buckets will share directory entries). As an example, Fig. 2 illustrates extendible hashing after inserting a new record with key 63 into Fig. 1. Bucket D overflows and the records in it are redistributed between D (where the last three bits of a record's hash value are 011) and D' (where the last three bits of a record's has value are 111). The directory doubles. The global depth is increased by one. The local depth of buckets D and D' are increased



Extendible Hashing. Figure 1. Illustration of the Extendible Hashing.



Extendible Hashing. Figure 2. The directory doubles after inserting 63 into Fig. 1.



Extendible Hashing. Figure 3. The directory does not double after inserting 17 and 13 into Fig. 2.

by one, while the local depth of the other buckets remains to be two. Except 111, which points to the new bucket D', each of the new directory entries points to the existing bucket which shares the last two bits. For instance, directory entry 101 points to the bucket referenced by directory entry 001.

In general, if the local depth of a bucket is d', the number of directory entries pointing to the bucket is $2^{d-d'}$. All these directory entries share the last d' bits.

To split an overflown bucket whose local depth is smaller than the global depth, one does not need to double the size of the directory. Instead, half of the $2^{d-d'}$ directory entries will point to the new bucket, and the local depth of both the overflown bucket and its split image are increased by one. For instance, Fig. 3 illustrates the extendible hashing after inserting 17 and 13 into Fig. 2. Bucket B overflows and a split image, bucket B', is created. There are two directory entries (001 and 101) that pointed to B before the split. Half of them (101) now points to the split image B'. The local depth of both buckets B and B' are increased by one.

Discussion

Deletion may cause a bucket to become empty, in which case it can be merged with its *buddy* bucket. The buddy bucket is referenced by the directory entry which shares the last (local depth − 1) bits. For instance, the buckets referenced by direction entries 1111 and 0111 are buddy buckets. Many deletions can cause the directory to halve its size and thus decrease its global depth. This is triggered by the bucket merging which causes all local depth to be strictly smaller than the global depth.

The fact that extendible hashing does not use any overflow page may significantly increase the directory size, as one insertion may cause the directory to double more than once. Consider the case when global depth is 3, and the bucket referenced by directory entry 001 overflows with five records 1, 17, 33, 49, 65. The directory is doubled. Directory entries 0001 and 1001 points to the overflown bucket and the split image. All the five keys will remain in the original bucket which is again overflowing. Therefore the directory has to be doubled again.

To alleviate this problem, one can allow a certain degree of overflow page links. For instance, whenever the fraction of buckets with overflow pages becomes larger than 1%, double the directory.

Key Applications

Extendible Hashing can be used in applications where exact match query is the most important query such as hash join [2].

Cross-references

- ▶ Bloom Filter
- ▶ Hashing
- ▶ Hash-based Indexing
- ▶ Linear Hashing

Recommended Reading

1. Fagin R., Nievergelt J., Pippenger N., and Strong H.R. Extendible hashing: a fast access method for dynamic files. ACM Trans. Database Syst., 4(3):315–344, 1979.
2. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 469–480.

Extensible Markup Language

► XML

eXtensible Stylesheet Language

► XSL/XSLT

eXtensible Stylesheet Language Transformations

► XSL/XSLT

Extensional Relational Database (ERDB)

► Decision Rule Mining in Rough Set Theory

External Hashing

► Hash-based Indexing

Extraction, Transformation, and Loading

Panos Vassiliadis[1], Alkis Simitsis[2]
[1]University of Ioannina, Ioannina, Greece
[2]IBM Almaden Research Center, San Jose, CA, USA

Synonyms

ETL; ETL process; ETL tool; Data warehouse back stage; Data warehouse refreshment

Definition

Extraction, transformation, and loading (ETL) processes are responsible for the operations taking place in the background of a data warehouse architecture. In a high level description of an ETL process, first, the data are extracted from the source data stores that can be online transaction processing (OLTP) or legacy systems, files under any format, web pages, various kinds of documents (e.g., spreadsheets and text documents) or even data coming in a streaming fashion. Typically, only the data that are different from the previous execution of an ETL process (newly inserted, updated, and deleted information) should be extracted from the sources. Secondly, the extracted data are propagated to a special-purpose area of the warehouse, called the data staging area (DSA), where their transformation, homogenization, and cleansing take place. The most frequently used transformations include filters and checks to ensure that the data propagated to the warehouse respect business rules and integrity constraints, as well as schema transformations that ensure that data fit the target data warehouse schema. Finally, the data are loaded to the central data warehouse (DW) and all its counterparts (e.g., data marts and views). In a traditional data warehouse setting, the ETL process periodically refreshes the data warehouse during idle or low-load, periods of its operation (e.g., every night) and has a specific time-window to complete. Nowadays, business necessities and demands require near real-time data warehouse refreshment and significant attention is drawn to this kind of technological advancement.

Historical Background

Despite the fact that ETL obtained its separate existence during the first decade of the twenty-first century, ETL processes have been a companion to database technology for a longer period of time, in fact, from the beginning of its existence. During that period, ETL software was just silently hidden as a routine programming task without any particular name or individual importance. ETL was born on the first day that a programmer constructed a program that takes records from a certain persistent file and populates or enriches another file with this information. Since then, any kind of data processing software that reshapes or filters records, calculates new values, and populates another data store than the original one is a form of an ETL program.

The earliest form of ETL system goes back to the EXPRESS system [13] that was intended to act as an

engine that produces data transformations given some data definition and conversion nonprocedural statements. In later years, during the early days of data integration, the driving force behind data integration were wrapper-mediator schemes; the construction of the wrappers is a primitive form of ETL scripting [12]. In the mid 1990s, data warehousing came in the central stage of database research and still, ETL was there, but hidden behind the lines. Popular books [3] do not mention ETL at all, although the different parts (transformation, cleansing, staging of intermediate data, and loading) are all covered – even if this is done very briefly at times (it is also noteworthy that the third edition of the same book in 2003 mentions ETL, although briefly). At the same time, early research literature treated data warehouses as collections of materialized views since this abstraction was simple and quite convenient for the formulation of research problems.

During the '00s, it became increasingly prevalent that ETL is really important for data integration tasks since it is costly, labor-intensive, and mission-critical – and for all these factors, important for the success of a data warehousing project. The difficulty lies in the combination of data integration and data cleaning tasks: as a typical integration problem, where data are moved and transferred from a certain data store to another, the details of schema and value mappings play a great role. At the same time, the data can be in highly irregular state (both in terms of duplicates, constraint, and business rule violations, and in terms of irregularities in the internal structure of fields – e.g., addresses). Therefore, finding effective ways for taming this nonregularity into a typical relational structure is very hard. Additionally, there are a number of persistent problems: ETL processes are hard to standardize, optimize, and execute in a failure-resilient manner (and thus the problem is hard to solve).

Foundations

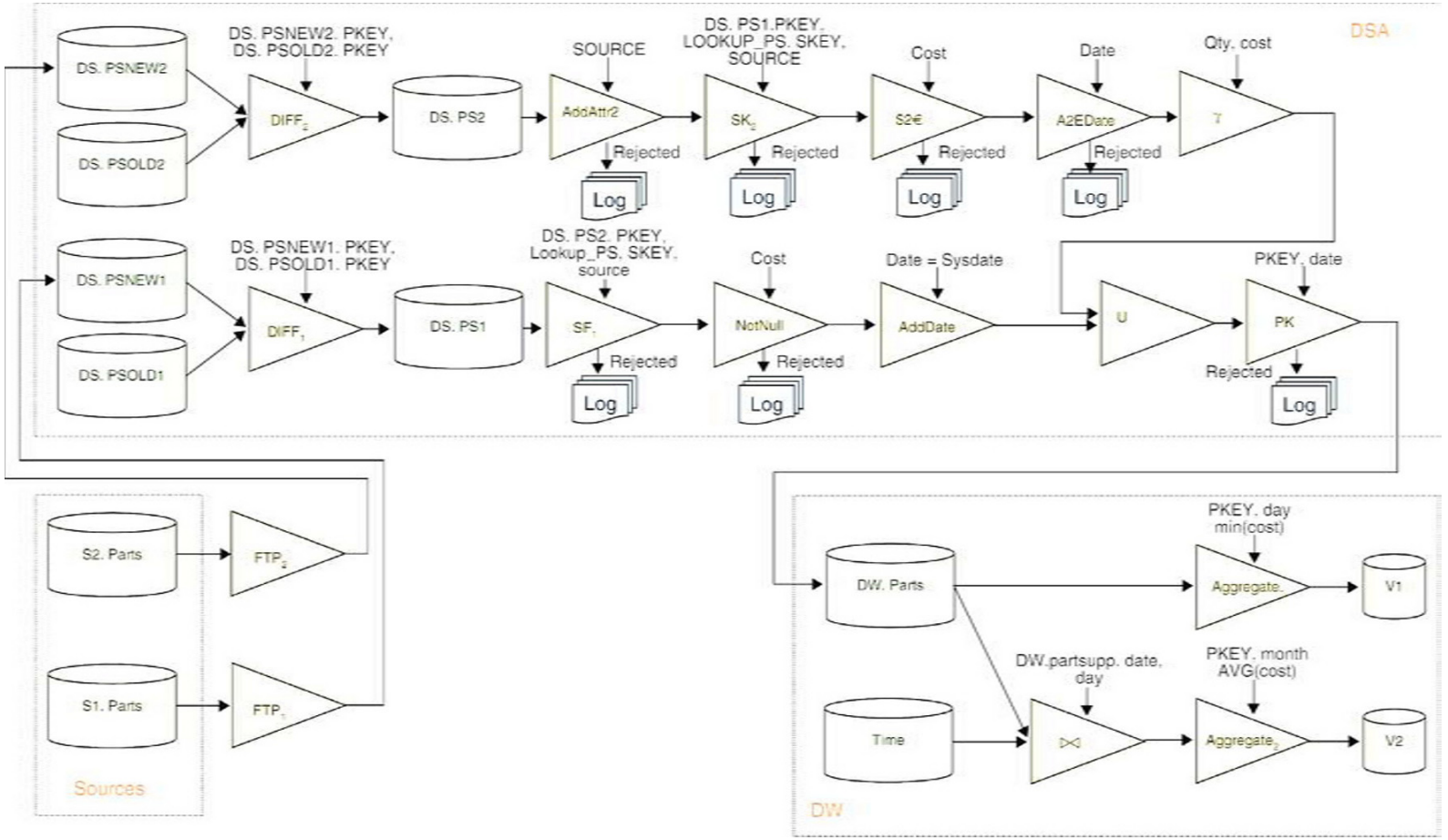
PART I. General description of an ETL process

Intuitively, an ETL process can be thought of as a directed acyclic graph, with activities and record sets forming the nodes of the graph, and input–output relationships between nodes forming its edges. Observe Figure 1, where two sources are depicted in the lower left part of the figure with the names $S_1.PARTS$ and $S_2.PARTS$. The data from these sources must be propagated to the target data warehouse fact table *DW.PARTS*, depicted at the bottom right part of the figure. Moreover, the newly inserted records must be further propagated to refresh aggregate views V_1 and V_2. (In fact, the views of the example can be anything among materialized views that are populated by the ETL process, materialized views automatically refreshed by the DBMS, convenient abstractions of data marts, reports, and spreadsheets that are placed in the enterprise portal for the end-users to download, and any other form of aggregated information that is published in some form to the end-users of the warehouse.) The whole process of populating the fact table and any of its views is facilitated by a *workflow of activities* that perform all the appropriate filtering, intermediate data staging, transformations and loadings. The upper part of Figure 1 depicts how the data (which are extracted as snapshots of the sources) are transported to a special purpose intermediate area of the warehouse, called the data staging area (DSA) [4], where the transformation phase takes place. First, the snapshots are compared with their previous versions, so that newly inserted or updated data are discovered. Secondly, these new data are stored in a hard disk so that in the case of a failure, the whole process should not start from scratch. Then, the data pass through several filters or transformations and they are ultimately loaded in the data warehouse. As mentioned, the warehouse fact or dimension tables are not necessarily the end of the journey for the data: at the bottom right of Figure 1, a couple of materialized views (standing as abstractions of reports, spreadsheets, or data marts for the sake of the example) are also refreshed with newly incoming data.

PART II. Individual steps

Extraction. The extraction step is conceptually the simplest task of all, with the goal of identifying the correct subset of source data that has to be submitted to the ETL workflow for further processing. As with the rest of the ETL process, extraction also takes place at idle times of the source system – typically at night. Practically, the task is of considerable difficulty, due to two technical constraints:

- The source should incur minimum overhead during the extraction, since other administrative activities also take place during that period.
- Both for technical and political reasons, administrators are quite reluctant to accept major



Extraction, Transformation, and Loading. Figure 1. An example ETL workflow.

interventions to their system's configuration; therefore, there must be minimum interference with the software configuration at the source side.

Depending on the technological infrastructure and the nature of the source system (relational database, COBOL file, spreadsheet, web site etc.) as well as the volume of the data that has to be processed, different policies can be adopted for the extraction step, which usually is also called "change data capture." The most naïve possibility involves extracting the whole source and processing it as if the original first loading of the warehouse was conducted. A better possibility involves the extraction of a snapshot of data, which is subsequently compared with the previous snapshot of data (either at the source, or the DSA side) and insertions, deletions, and updates are detected. In this case, there is no need to further process the data that remain the same. The work presented in [5] is particularly relevant in this context. Another possibility involves the usage of triggers in the source that are activated whenever a modification takes place in the source database. Obviously, this can be done only if the source database is a relational system; most importantly though, both the interference with the source system and the runtime overhead incurred are deterring factors with respect to this option. An interesting possibility, though, involves "log sniffing," i.e., the appropriate parsing of the log file of the source. In this case, all modifications of committed transactions are detected and they can be "replayed" at the warehouse side.

A final point in the extraction step involves the necessity of encrypting and compressing the data that are transferred from the source to the warehouse, for security and network performance reasons, respectively.

Transformation. Depending on the application and the tool used, ETL processes may contain a plethora of transformations. In general, the transformation and cleaning tasks deal with classes of conflicts and problems that can be distinguished at two levels [7]: the schema and the instance level. In this article, a broader classification of the problems is presented that includes value-level problems, too.

- *Schema-level problems.* The main problems with respect to the schema level are (a) naming conflicts, where the same name is used for different objects (homonyms) or different names are used for the same object (synonyms), and (b) structural conflicts, where one must deal with different representations of the same object in different sources, or converting data types between sources and the warehouse.
- *Record-level problems.* The most typical problems at the record level concern duplicated or contradicting records. Furthermore, consistency problems concerning the granularity or timeliness of data occur, since the designer is faced with the problem of integrating data sets with different aggregation levels (e.g., sales per day vs. sales per year) or reference to different points in time (e.g., current sales as of yesterday for a certain source vs. as of last month for another source).
- *Value-level problems.* Finally, numerous low-level technical problems may be observed in different ETL scenarios. To mention a few, there may exist problems in applying format masks, like for example, different value representations (e.g., for sex: "Male," "M," "1"), or different interpretation of the values (e.g., date formats: American "mm/dd/yy" vs. European "dd/mm/yy"). Other value-level problems include assigning surrogate key management, substituting constants, setting values to NULL or DEFAULT based on a condition, or using frequent SQL operators like UPPER, TRUNC, and SUBSTR.

To deal with such issues, the integration and transformation tasks involve a wide variety of functions, such as normalizing, denormalizing, reformatting, recalculating, summarizing, merging data from multiple sources, modifying key structures, adding an element of time, identifying default values, supplying decision commands to choose between multiple sources, and so forth.

Loading. The end of the source records' journey through the ETL workflow comes with their loading to the appropriate table. A typical dilemma faced by inexperienced developers concerns the choice between bulk loading data through a DBMS-specific utility or inserting data as a sequence of rows. Clear performance reasons strongly suggest the former solution, due to the overheads of the parsing of the insert statements, the maintenance of logs and rollback-segments (or, the risks of their deactivation in the case of failures). A second issue has to do with the possibility of efficiently discriminating records that are to be inserted for the first time, from records that act as updates to previously loaded data. DBMSs typically support some

declarative way to deal with this problem (e.g., Oracle's MERGE command [9]). In addition, simple SQL commands are not sufficient since the "open-loop-fetch" technique, where records are inserted one by one, is extremely slow for the vast volume of data to be loaded in the warehouse. A third performance issue that has to be taken into consideration has to do with the existence of indexes, materialized views, or both, defined over the warehouse relations. Every update to these relations automatically incurs the overhead of maintaining the indexes and the materialized views.

PART III. Global picture revisited

Design and Modeling. After the above analysis, one can understand that the design of ETL processes is crucial and complex at the same time. There exist several difficulties underlying the physical procedures already described.

At the beginning of a data warehousing project, the design team in cooperation with the business managers have to clarify the requirements, identify the sources and the target, and determine the appropriate transformations for that specific project; i.e., the first goal is to construct a *conceptual design of the ETL process* [8,18,20]. The most important part of this task is to identify the schema mappings between the source and the target schemata. This procedure is usually done through analysis of the structure and content of the existing data sources and their intentional mapping to the common data warehouse model. Possible data cleaning problems can also be detected at this early part of the design. Conceptual modeling formalisms (both ad- hoc and UML-based) have been proposed in the literature [8,18,20]. In addition, several research approaches have been proposed toward the automation of the schema mapping procedure [10]; one of the most prominent examples is CLIO, which has also been adapted by commercial solutions proposed by IBM [2]. However, so far, the automatic schema mapping supports cases of simple ETL transformations. On the contrary, there exists a semi-automatic method that uses ontologies and semantic web technology to infer the appropriate interattribute mappings along with the respective transformations needed in an ETL process [16].

When this task ends, the design involves the *specification of a primary data flow* that describes the route of data from the sources toward the data warehouse, as they pass through the transformations of the workflow. The execution sequence of the workflow's transformation is determined at this point. The data flow defines what each process does and the execution plan defines in which order and combination. The flow for the logical exceptions – either integrity or business rules violations is specified, too. Control flow operations that take care of monitoring or failure occurrences can also be specified. Most ETL tools provide the designer with the functionality to construct both the data and the control flow for an ETL process.

Optimization. Usually, an ETL workflow must be completed in a specific time window and this task is realized periodically; e.g., each night. In the case of a failure, the quick recovery of the workflow is also important [6]. Hence, for performance reasons, it is necessary to optimize the workflow's execution time. Currently, although the leading commercial tools provide advanced GUI's for the design of ETL scenarios, they do not support the designer with any technique to optimize the created scenarios. Unlike relational querying, where the user declaratively expresses a query in a high-level language and the DBMS optimizer decides the physical execution of the query automatically, in the case of ETL workflows it is the designer who must decide the order and physical implementation for the individual activities.

Practical alternatives involve (a) letting the involved DBMS (in the case of DBMS-based scripts) do the optimization and (b) treating the workflow as a big multi-query. The solution where optimization is handed over to the DBMS for execution is simply not sufficient, since DBMS optimizers can interfere only in portions of a scenario and not in its entirety. Concerning the latter solution, it should be stressed that *ETL workflows are not big queries*, since:

- It is not possible to express all ETL operations in terms of relational algebra and then optimize the resulting expression as usual. In addition, the cases of functions with unknown semantics – "black-box" operations – or with "locked" functionality – e.g., an external call to a DLL library – are quite common.
- Failures are a critical danger for an ETL workflow. The staging of intermediate results is often imposed by the need to resume a failed workflow as quickly as possible.
- ETL workflows may involve processes running in separate environments, usually not simultaneously and under time constraints; thus their cost

estimation in typical relational optimization terms is probably too simplistic.

All the aforementioned reasons can be summarized by mentioning that neither the semantics of the workflow can always be specified, nor its structure can be determined solely on these semantics; at the same time, the research community has not come-up with an accurate cost model so far. Hence, it is more realistic to consider ETL workflows as complex transactions rather than as complex queries. Despite the significance of such optimization, so far the problem has not been extensively considered in research literature, with results mainly focused on the black-box optimization at the logical level, concerning the order with which the activities are placed in the ETL workflow [14,15].

Implementation and Execution. An ETL workflow can either be hand-coded, or specified and executed via an ETL tool. The hand-coded implementation is a frequent choice, both due to the cost of ETL tools and due to the fact that developers feel comfortable to implement the scripts that manipulate their data by themselves. Typically, such an ETL workflow is built as the combination of scripts written in some procedural languages with high execution speed (for example, C or Perl) or some vendor specific database language (PL\SQL, T-SQL, and so on). Alternatively, ETL tools are employed, mainly due to the graphical programming interfaces they provide as well as for their reporting, monitoring, and recovery facilities.

Key Applications

ETL processes constitute the backbone of the data warehouse architecture. The population, maintenance, evolution, and freshness of the warehouse depends heavily on its backstage where all the ETL operations are taken place. Hence, in a corporate environment there is a necessity for a team devoted to the design and maintenance of the ETL functionality.

However, ETL is not useful only for the refreshment of large data warehouses. Nowadays, with the advent of Web 2.0 new applications have emerged. Among them, mashups are web applications that integrate data that are dynamically obtained via web-service invocations to more than one sources into an integrated experience. Example applications include Yahoo Pipes (http://pipes.yahoo.com/), Google Maps (http://maps.google.com/), IBM Damia (http://services.alphaworks.ibm.com/damia/), and Microsoft Popfly (http://www.popfly.com/). Under the hood, the philosophy for their operation is "pure" ETL. Although, the extraction phase mainly contains functionality that allows the communication over the web, the transformations that constitute the main flow resemble those which are already built-in in most ETL tools. Different applications are targeting to gather data from different users, probably in different formats, and try to integrate them into a common repository of datasets; an example application is the Swivel (http://www.swivel.com/).

Future Directions

Although the ETL logic is not novel in computer science, several issues still remain open. A main open problem in the so called traditional ETL is the agreement upon a unified algebra and/or a declarative language for the formal description of ETL processes. However, there are some first results in the context of the data exchange problem [1]. As already mentioned, the *optimization* of the whole ETL process and of any individual transformation operators pose interesting research problems. In this context, parallel processing of ETL processes is of particular importance. Finally, *standardization* is a problem that needs an extra note of attention. The convergence toward a globally accepted paradigm of thinking and educating computer scientists on the topic is a clear issue for the academic community.

However, the ETL functionality expands into new areas beyond the traditional data warehouse environment, where the ETL is executed off-line, on a regular basis. Such cases include but are not limited to: (a) *On-Demand ETL processes* that are executed sporadically (typically for Web data), and they are manually initiated by some user demand [11]; (b) *Stream ETL* that involves the possible filtering, value conversion, and transformations of incoming streaming information in a relational format [11]; (c) *(near)Real-Time ETL* that captures the need for a data warehouse containing data as fresh as possible.

Finally, with the evolution of the technology and the broader use of internet from bigger masses of users, the interest is moved also *to multiple types of data*, which do not necessarily follow the traditional relational format. Thus, modern ETL applications should also handle novel kinds of data, like XML, spatial, biomedical or multimedia data efficiently.

Data Sets

Benchmarking the ETL process is a clear problem nowadays (2007). The lack of any standard, principled experimental methodology with a clear treatment of the workflow complexity, the data volume, the amount of necessary cleaning, and the computational cost of individual activities of the ETL workflows is striking. The only available guidelines for performing a narrow set of experiments are given in the TPC-DS standard [17], and the first publicly available benchmark for ETL processes is presented in [19].

Cross-references

- ▶ Active Data Warehousing
- ▶ Data Cleaning
- ▶ Data Warehouse
- ▶ Multidimensional Modeling
- ▶ (near)Real-Time Data Warehousing

Recommended Reading

1. Fagin R., Kolaitis P.G., and Popa L. Data exchange: getting to the core. ACM Trans. Database Syst., 30(1):174–210, 2005.
2. Haas L.M., Hernández M.A., Ho H., Popa L., and Roth M. Clio grows up: from research prototype to industrial tool. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 805–810.
3. Inmon W. Building the Data Warehouse, 2nd edn. Wiley, New York, 1996.
4. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, New York, 1998.
5. Labio W. and Garcia-Molina H. Efficient snapshot differential algorithms for data warehousing. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 63–74.
6. Labio W., Wiener J.L., Garcia-Molina H., and Gorelik V. Efficient resumption of interrupted warehouse loads. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 46–57.
7. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
8. Luján-Mora S., Vassiliadis P., and Trujillo J. Data Mapping Diagrams for Data Warehouse Design with UML. In Proc. 23rd Int. Conf. on Conceptual Modeling, 2004, pp. 191–204.
9. Oracle , Oracle 9i SQL Reference. Release 9.2. 2002.
10. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
11. Rizzi S., Abelló A., Lechtenbörger J., and Trujillo J. Research in data warehouse modeling and design: dead or alive? In Proc. 9th ACM Int. Workshop on Data Warehousing and OLAP, 2006, pp. 3–10.
12. Roth M.T. and Schwarz P.M. Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 266–275.
13. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: a data extraction, processing, and restructuring system. ACM Trans. Database Syst., 2(2):134–174, 1977.
14. Simitsis A., Vassiliadis P., and Sellis T.K. Optimizing ETL processes in data warehouses. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 564–575.
15. Simitsis A., Vassiliadis P., and Sellis T.K. State-space optimization of ETL workflows. IEEE Trans. Knowl. Data Eng., 17 (10):1404–1419, 2005.
16. Skoutas D. and Simitsis A. Designing ETL processes using semantic web technologies. In Proc. 9th ACM Int. Workshop on Data Warehousing and OLAP, 2006, pp. 67–74.
17. TPC, TPC-DS (Decision Support) specification, draft version 52. 2007.
18. Trujillo J. and Luján-Mora S. A UML based approach for modeling ETL processes in data warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003, pp. 307–320.
19. Vassiliadis P., and Karagiannis A., and Tziovara V., and Simitsis A. Towards a Benchmark for ETL Workflows. In Proc. 5th Int. Workshop on Quality in Databases, 2007, pp. 49–60.
20. Vassiliadis P., Simitsis A., and Skiadopoulos S. Conceptual modeling for ETL processes. In Proc. 5th ACM Int. Workshop on Data Warehousing and OLAP, 2002, pp. 14–21.

Extrinsic Time

- ▶ Transaction Time