Architectural Patterns for Data Mediation in Web-centric Information Systems

Jens H. Jahnke, Daniel M. German
Department of Computer Science
University of Victoria
PO Box 3055, Victoria B.C., Canada V8W3P6
[jens|dmgerman]@cs.uvic.ca

Jörg P. Wadsack[1]
Department of Mathematics and Computer Science
University of Paderborn
Warburger Str. 100, 33098 Paderborn, Germany
maroc@upb.de

Abstract

Web-centric information management has considerably changed business processes of organisations in the private and public sector. The Web itself has emerged from a fairly static collection of hypertext documents to a dynamic network of objects providing information services, i.e., the so-called Object-Web [OHE99]. Today, many organisations provide access to their information systems (IS) by means of Web services. Moreover, the Web is no longer restricted to traditional computing hardware but it invades the world of mobile and smart devices. We can identify three main waves in the evolution of Web-centric systems. The first wave had its peak in the mid 90's; Organisations discovered the Web as an environment for marketing their products and publishing information to a growing community of users. The second, still ongoing wave has been focused on business-to-business e-commerce and federated Web services. The third wave of connectivity is on the horizon and targets the integration of "every-day-things" like cell phones, PDAs, and household appliances as Web clients based on emerging W3C standards like UDDI and SOAP. In this paper, we discuss a collection of general architectural patterns that can be applied to systematically building such "third generation" Web information systems. We then specialize these patterns with respect to current middleware solutions applied to a real world case study of a third generation Web IS in the domain of Health Care.

Keywords
Web engineering, design patterns, component-based software development, connection-based programming, information mediation, architectural evolution

1. Introduction

For over two decades, Web-centric information management has considerably changed business processes of organisations in the private and public sector. The Web itself has emerged from a fairly static collection of hypertext documents to a dynamic network of objects providing information services, i.e., the so-called *Object-Web* [OHE99]. Today, many organisations provide access to their information systems (IS) by means of Web services. Moreover, the Web is no longer restricted to traditional computing hardware but it invades the world of mobile and smart devices. We can identify three main waves in the evolution of Web-centric systems. The first wave had its peak in the 90's: organisations discovered the Web as an environment for marketing their products and publishing information to a growing community of users. The second, still ongoing, wave has been focused on *business-to-business* e-commerce and federated Web services The third wave of connectivity is on the horizon and targets the integration of "every-day-things" like cell phones, PDAs, and household appliances as Web clients based on emerging W3C standards like UDDI and SOAP.

It is widely anticipated that this third wave of Web-centric information exchange will have much greater impact on our societies than the recent two. The main reason for this prediction is that the third wave of connectivity will include electronic information management that works "under the hood" and involves almost everybody—not only those working in particular functions. Our experience with several industrial case studies clearly indicates that future Web-centric systems have to integrate the Web in the form of all three waves in order to be successful and competitive. In this document, we refer to Web-centric systems that integrate technologies from all three waves of connectivity as *third generation* systems.

Information gathering, information publishing and information mediation are common to all IS components participating in third generation systems. In this paper, we discuss a collection of general architectural patterns that can be applied to systematically building third generation systems. We then specialize these patterns with

[1] Jörg Wadsack has been a visiting researcher at the University of Victoria from January to April 2002

respect to current middleware solutions applied to a real world case study of a third generation Web IS in the area of Health Care.

In the next section, we introduce the concept of architectural patterns in general and applied to Web-centric IS in particular. In Section 3, we propose a pattern collection for Web-centric IS. Section 4 discusses the application of these patterns to a case study in the health care sector. We then relate our work to other research efforts in this area and close with our conclusions and remarks about future work.

2. Architecture of Web-centric Information Systems

Modern "third generation" Web IS deal with large amounts of information spread over multiple locations and heterogeneous platforms. Different kinds of interfaces are needed to gather, publish or mediate information for an organisation, among organisations and between organisational IS and mobile devices.

Figure 1 illustrates the architecture of a typical organisation whose data is divided in different federated databases. The organisation makes selected parts of this information available to clients by means of Web browsers or to other organisations for the purpose of electronic data interchange. Information from different organisation may also be published in an aggregate form, e.g., tickets offered by different airlines. This aggregation can explicitly be visible to the client, e.g., different frames in a Web browser, or transparent to the client, i.e., information is merged and then published and looks as it stems from one single source.

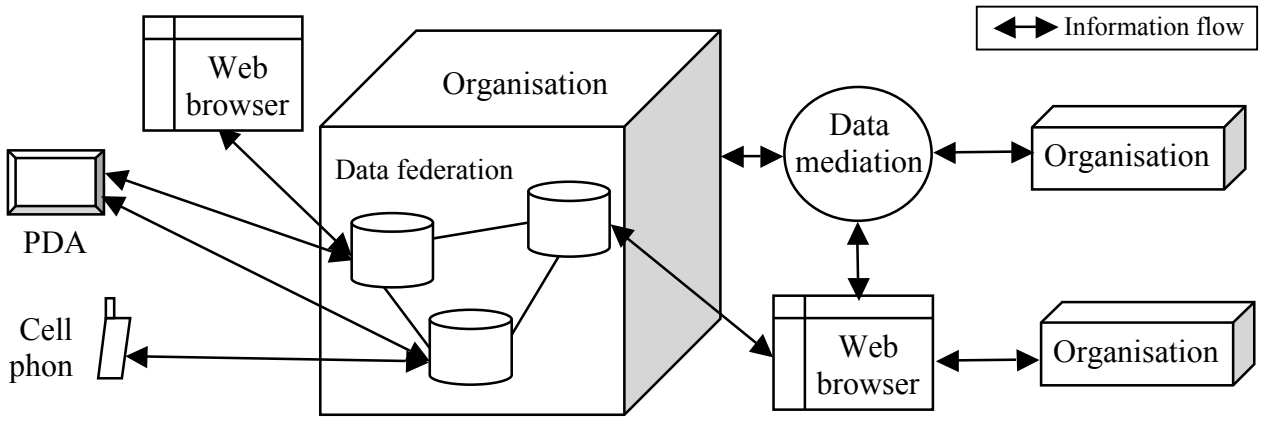

Figure 1 Third generation Web-centric information system

Traditional architectures for Web-centric information systems are based on the procedure-call paradigm, i.e., clients call service operations on server objects. This traditional development paradigm implies that the client programmer knows about the servers at the development time of the client software. This is disadvantageous in case of rapidly evolving architectures such as third generation Web IS. Therefore, software engineers have recently started to migrate to a new paradigm called component-oriented software development.

The component-oriented software development paradigm promotes connection-based programming. This means that Web components are defined with well-defined interfaces in partial ignorance of each other. The actual connections among components are instantiated and deployed later. In other words, connections are now treated as "active" first-order citizens in distributed architectures [Szy98]. This supports networked evolution because it facilitates adding and removing connections with little changes to the components in a system. The patterns described in this paper are component-based and promote connection-based programming for distributed third generation Web IS.

In general, architectural patterns define the responsibility of typical parts of a system. Furthermore, they provide rules and guidelines for relationships between those components. Architectural patterns express and separate the concerns of fundamental structures in software systems [BMR+96].

We present four architectural patterns in this paper:

- *Data Portal*
- *Data Connection*
- *Data Fusion*
- *Data Transducer*

Their relationships are depicted in Figure 2, and in the next section we describe these architectural patterns in detail. Distributed information management components are connected to the Web by means of the *Data Portal* pattern. The Data Portal is just one of three general *Data Component* patterns that can be connected through instances of the *Data Connection* pattern. The *Data Fusion* pattern is used to merge information from separate sources. Finally, the *Data Transducer* translates information into a different structure. This pattern is used for mediating among different data representations, as well as for rendering Web data for presentation to human clients (e.g., in browsers).

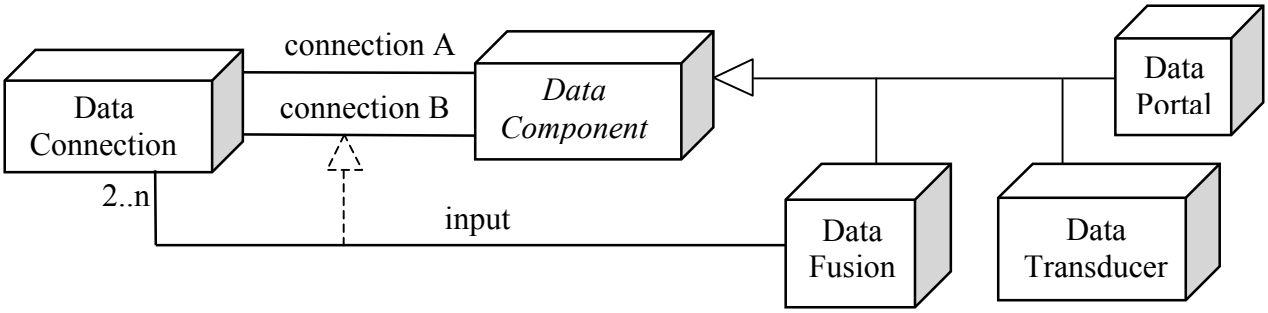

Figure 2 Architectural patterns: overview

3. Architectural patterns for Web Data Management

In this section, we refine the description of the four patterns named in the previous section.

3.1. Data Portal

Name: **Data Portal, (Export/Import)**

Intent: A **data portal** is an interface between the world and the IS subsystems of an organisational entity. Its purpose is to make selected parts of the data (maintained within this entity) accessible for authorized external clients (services or users). There are two different versions of this pattern: ***Export Data Portal***, which is responsible for making internal data available outside the organisation; and ***Import Data Portal***, which is used to import information from external sources into the organisational databases. In practical applications, a clear distinction between these two patterns might not always be possible. The combination of exporting internal data and importing external data is an ***Export & Import Data Portal***.

Motivation: Three main motivations exist for the **data portal**, namely resolving *heterogeneity*, providing *data security*, and increasing *data availability*. Heterogeneity reflects on the fact that different organisational entities utilise various heterogeneous platforms and technologies for their data repositories. Interoperability between different organisational entities requires that this heterogeneity be resolved. The Data portal serves this purpose by exploiting Web-interoperability standards. Second, the requirement for data security stems from the fact that, in many cases, external clients should not have access to all the data stored in internal databases (e.g., for protecting intellectual property, personal or sensitive information, etc). The **data portal** provides a level of isolation between the effective schema (as it is perceived by the external client), and the source schemas of the internal databases in question. Finally, the data portal facilitates fast access to a unified view of internal data structures. This is needed because data is often distributed among various different transactional and analytical repositories within organisations. The **data portal** serves as a façade for these various data sources, in which the effective schema is a buffered view of the collection of schemas of all the involved databases.

Applicability: The **data portal** pattern can be used whenever the organisation needs to make available part of its data to entities outside its intranet. The **export data portal** specializes in allowing external entities to query the internal databases, while the **import data portal** is used to update the internal databases with data from the outside of the organisation.

Structure: Figure 3 shows the structure of the pattern.

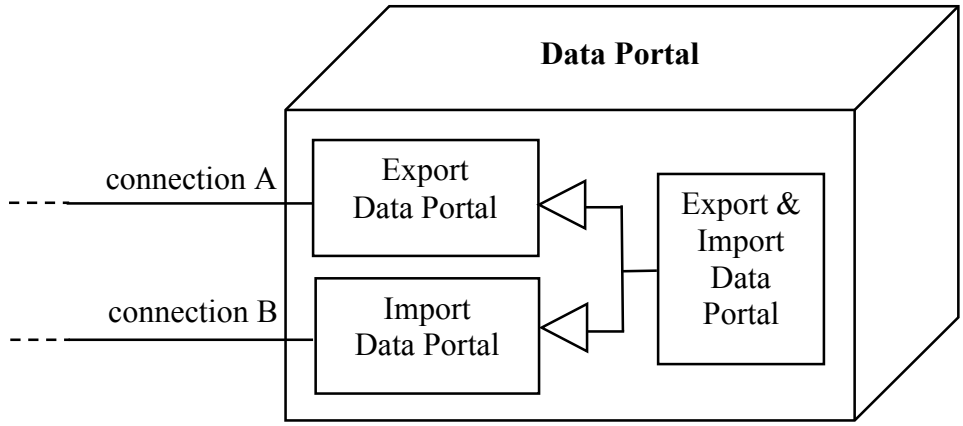

Figure 3 Structure of Data Portal pattern

Participants:

- *Source schemas.* The schemas of the different data sources that compose the information systems of the organisation.
- *External schema.* The schema of the organisation data as seen by the external clients.
- *Mapping function.* A function that converts the source data (from the different source data sources and each conforming to a different source schema) to the effective schema.

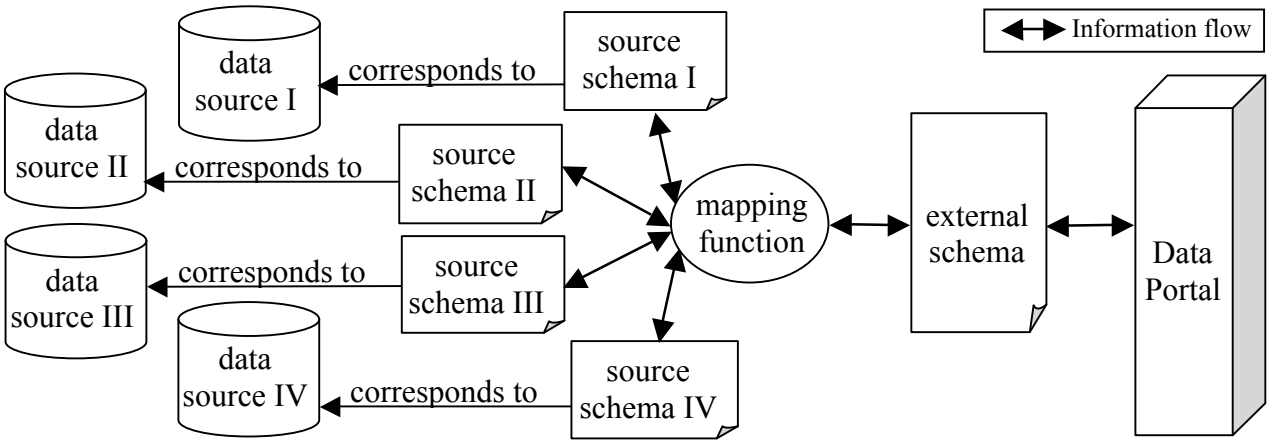

Figure 4 Inner-organisational view of Data Portal pattern

Collaborations: Each of the internal data sources is described with its own schema (a *source schema).* An external client, however, is not expected to be able to see any of these data sources. Instead, the client has access only to the "view" that the organisation allows. This view is described using the *external schema*. The *mapping function* is responsible for resolving a request from the client (based on the *external schema*) into a request to the internal databases (described using the *source schemas)*

Consequences: There is a single entry point to external access and updates to the internal databases of the organisation. From the point of view of the external client, there is a single schema that corresponds to a single data source. The client does not have to worry about the different data sources, their types (whether they are ODBC, file-based, or other) nor how to query or update them individually. The organisation, on the other hand, can filter and block access to sensitive information in the source databases. A disadvantage is the complexity of building this federative layer is significantly higher than using direct access, e.g., by using JDBC.

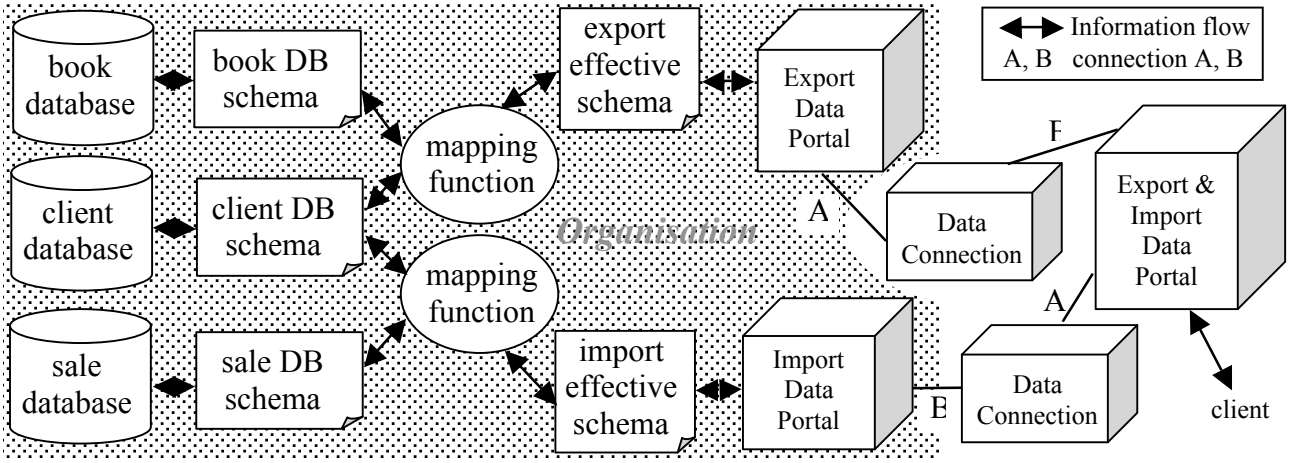

Figure 5 Data Portal pattern examples

3.2 Data Connection

Name: **Data Connection**
Intent: Proactive service for pulling data of interest from a given data source and pushing this data to a given data sink.
Motivation: Traditionally, data exchange has been controlled by either the client or the server of a distributed IS. As motivated earlier in this paper, this concept has limited scalability and flexibility for the rapidly evolving distributed architectures of third generation Web IS. Therefore, Web engineers have begun to treat data connections as first-order citizens in their architectural designs. This trend is merely reflecting on a general trend in current software engineering practice from the traditional call-procedure paradigm to the new paradigm of connection-based programming [Szy98]. For the application domain of engineering distributed Web IS, a Data Connection is an instance that controls the exchange of electronic data among several component IS.
Applicability: Whenever we need to constantly retrieve information from a data source.
Structure: Figure 6 shows the structure of the pattern.

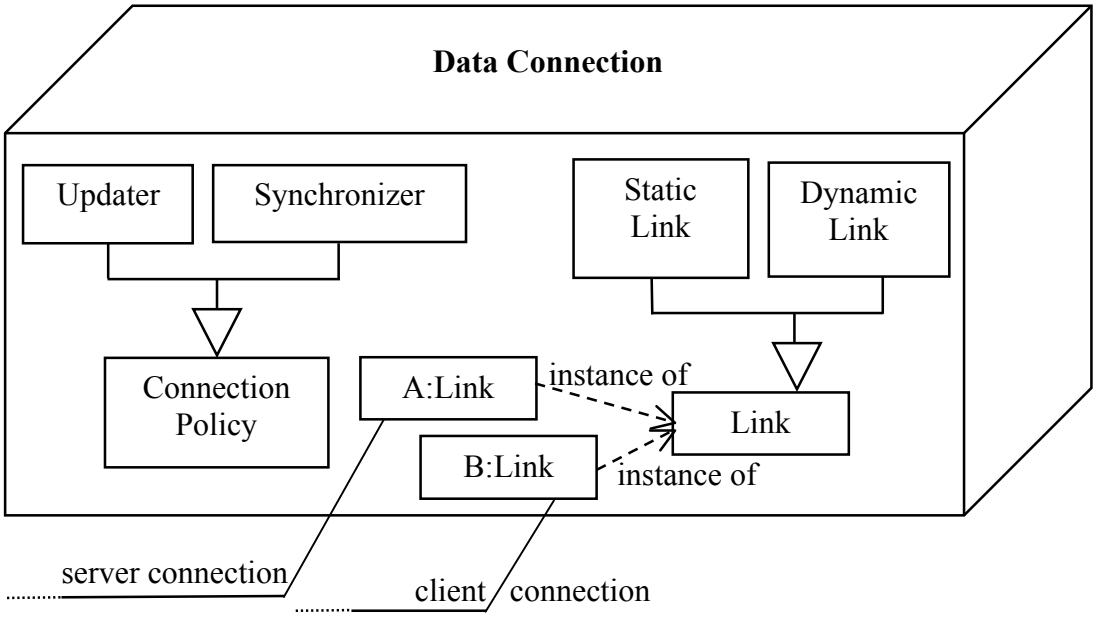

Figure 6 Data Connection pattern

Participants:

- *Server and Client Connections*. These connections are created from the instance of the **data connection** pattern to the server and the client, and can be either static or dynamic links. A static link assumes a reliable connection between the client and the server, while a dynamic link can handle a change in the IP address of the client and temporal disconnection between client and server (both situations common in mobile devices).
- *Connection Policy*. Indicates the properties of the data connection. An *updater data connection* is only concerned with handling continuous updates to the client (stock market prices, news updates), while the *synchronizer* assumes that the client has a copy of the data and needs to synchronize it with the master copy in the server.
- *Server*. The Web service to which the client wants to connect.
- *Client*. The client's Web IS, that needs to connect to the Web Service.

Collaborations: When instantiated, the **data connection** pattern creates communication links to both, the *client* and the *server*. The instance of the **data connection** pattern, according to its *connection policy*, receives requests from the *client* and converts them into requests to the *server*. The reply from the *server* is then translated and sent to the *client*. The **data connection** is responsible for the issues related to the connection to the service, such as authentication, encryption, roaming, temporal disconnection, etc.

Consequences: The main benefit of connection-based programming is *late binding* of component IS. This means that it becomes possible to develop component IS with well-defined interfaces (data portals) more-or-less in isolation from each other, and flexibly connect them at a later point in time. The **data connection** pattern is also responsible for handling the complexities of the communication with the service, allowing the client to be unaware of them.

3.3 Data Fusion

Name: **Data Fusion**

Intent: To combine the data received from two or more data sources, into a single, unified data source.

Motivation: One of the goals of the semantic Web (as defined by the W3C) is the availability of a variety of data sources. These data sources will be mined by intelligent agents, which will understand their schemas and then manipulate and combine their data, and then present it to the client as a unified data source. The client does not need to deal with these complexities, and instead, can assume a unique data source.

Applicability: A distributed Web-IS often must combine and aggregate data from several sources into a common data stream. A travel agency Web-IS, for example, requires access to all the different airlines data sources, in order to present all the available flights to the potential traveller.

Structure: Figure 7 shows the structure of the pattern.

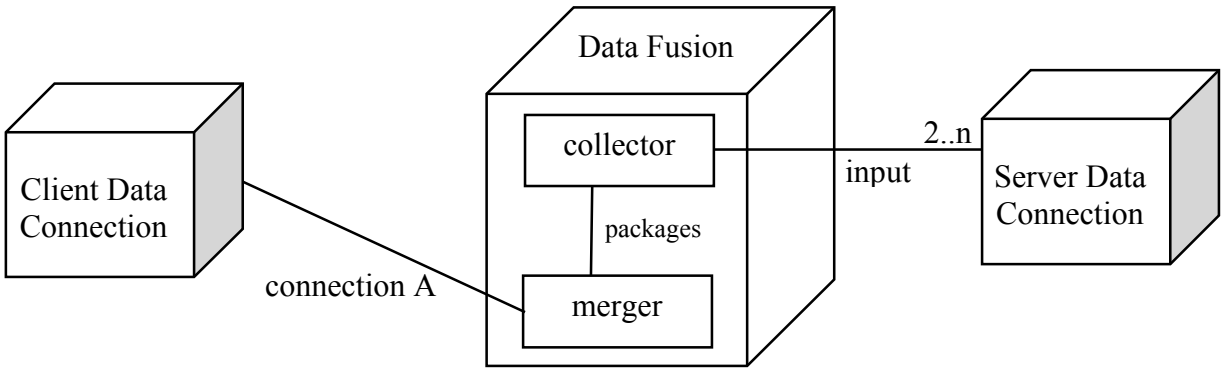

Figure 7 Data Fusion pattern

Participants:

- *Server data connections*. A data connection is created to connect the data fusion to each of the servers.
- *Client data connection*. The data connection to the client.
- *Collector*. Queries the different servers and receives their results.
- *Merger*. It is responsible for merging the results from the different data sources.

Collaborations: The instance of the **data fusion** pattern creates *data connections* to each of the servers. When the instance of the pattern receives a query, it translates it into a sequence of queries, each intended for a different server. The *collector* is then responsible for executing these queries (using a *data connection* to the corresponding server). The *collector* receives the results and passes them to the *merger*, who proceeds to combine them. The resulting data is then sent to the client using the *client data connection.*

Consequences: The availability of information in XML from different Web services requires the existence of **data fusion** instances that merge these sources info a unified data source. For example, combining air flights, hotels and car rental reservations, for a travel agent Web-IS. A client application that uses a **data fusion** pattern does not need to worry about the complexities of accessing and merging multiple data sources.

3.4 Data Transducer

Name: **Data Transducer**

Intent: To convert data in a given source format to data in another target format.

Motivation: There are cases where the client might not be able to interpret the data in its original format or schema. In that case, the data **transducer** pattern converts the source data into a target format that the client expects.

Applicability: Whenever an application requires data in a different format than the one that the data source provides. A client application might be designed with a given schema in mind, but the server might provide data in a different schema. Transducing the data from one schema to another will allow both applications to interact without changes in either one of them. For example, the client expects data in a different DTD than the source produces.

Structure: Figure 8 depicts the structure of the pattern.

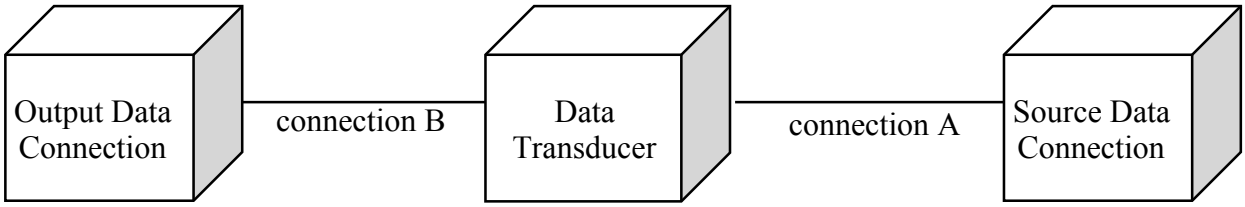

Figure 8 Data Transducer pattern

Participants:

- *Source data connection.* A data connection to the source of the data.
- *Output data connection.* A data connection to the consumer of the data.
- *Source schema.* The schema of the original source data.
- *Result schema.* The desired schema for the resulting data.
- *Transducing function.* A function that converts the data from the *source schema* into the *result schema.*

Collaborations: The producer of data is connected to the data **transducer** using a *source data connection;* similarly, the consumer of the data is connected to the data **transducer** using an *output data connection.* The *source data connection* pulls the data to be transduced. This data conforms to the *source schema* and it is used as input to the *transducing function.* The output data, which conforms to the *result schema,* is then fed to the *output data connection.*

Consequences: This pattern allows the interaction of two applications, designed for different schemas, to interoperate. The two applications do not need to be aware that the schema of the other is different. The disadvantage of this pattern is that the transduction could lose information because the destination schema might not be able to convey all the information of the source schema.

4. Case Study: Palliative Care Web

In this section, we describe applications of the patterns introduced above to an example case study of a third generation Web IS. This case study is carried out in a collaborative effort of the Department of Computer Science at the University of Victoria B.C. (UVic) with the UVic School for Health Information Science and other medical centers, including a local cancer clinic. The subject of the collaborative project is building a third generation Web information system in the area of Palliative Care[2]. The background for this project has been the ongoing initiative of Health Canada to establish an integrated surveillance system for key service areas in medical care, particularly end-of-life care. Such a system would collect data about the clinical practice in different medical centers, e.g., hospices, cancer clinics, etc. The collection of data is done with PDAs (Victoria) or handheld input tablets. The heterogeneous data sets at the different locations would then be consolidated with respect a standard reference structure. This reference structure is called the Palliative Data Set (PDS) and is currently refined by a joint group of Health researchers in Victoria and Toronto [Kuz02]. The consolidated and combined data would then be stored in a Data Warehouse at the Ministry of Health, where it can be browsed and analysed for the purpose of knowledge discovery and surveillance.

Architecture

The following figure shows the main components of the *Palliative Care Web* architecture. The architecture integrates three main organisational sites, namely in Victoria, Toronto and Edmonton. Each organisational site has its own proprietary IS infrastructure in terms of different kinds of databases and repositories. Two portals per site hide the complexity and heterogeneity of these idiosyncratic infrastructures. For increased clarity, we refer to these portals as *ubiquitous portal* and *mediation portal*, respectively. The ubiquitous portal interfaces to handheld and embedded devices for the purpose of viewing and collecting data from clinical practice. This interface is realized by means of a *Data Connection* and *Data Transducer* pattern (for rendering clinical data on the embedded device. Both pattern instances reside on the embedded device. The link from the *Data Connection* to the ubiquitous portal is a dynamic link and the connection policy is a *Synchronizer* since the embedded handheld devices can be used off-line (see the *Data Connection* pattern).

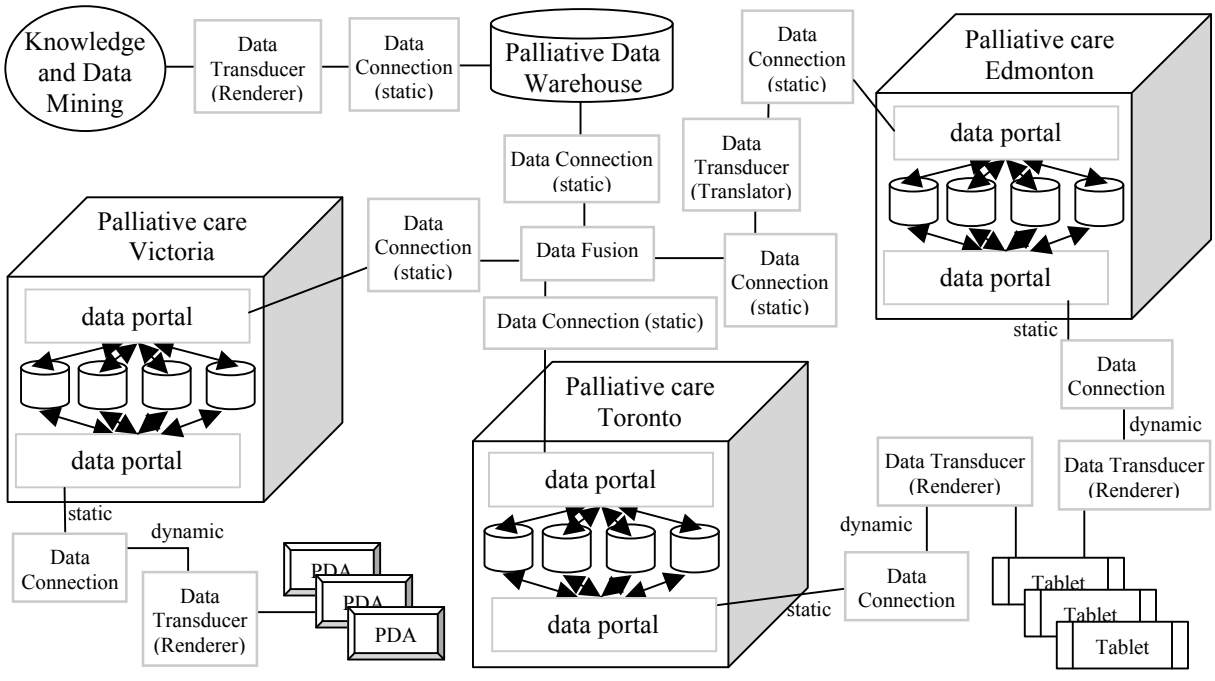

Figure 9 Palliative Care Web

Furthermore, Figure 9 shows that the *Data Fusion* pattern is used for combining the palliative data sets from the different organisational sites. In cases where data sets are not structurally compliant to the standard PDS format, a *Data Transducer* pattern is instantiated to provide the structural translation. All instances of Data Connection deployed between the Palliative Warehouse and the organisational sites enact an *Update* policy, because information is communicated only in one direction (to the Palliative Warehouse).

[2] Palliative care deals with the treatment of pain and the prescription of pain relieving drugs.

Implementation Details

The current implementation status of the described case study is as follows. The ubiquitous portals are in place at the organisational sites. The mediation portals are not yet in place at the actual organisational sites. However, the joint UVic group (composed of the Computer Science and the Health Information Science departments) has set up distributed Web servers that simulate these organisational sites. The *Data Portal* (mediation portal) has been implemented based on IBM Alphaworks XML Lightweight Extractor (XLE) [XLE]. XLE can query any ODBC data sources and emit the result in XML. How XLE works can be customised by an annotated DTD that defines the mapping between ODBC data and the XML representation. Web access (via HTTP) is provided using a Tomcat Apache Web server. This service allows instances of the Data Connection patterns to pull XML data by using the Simple Object Access Protocol (SOAP). *Data Transducer* and *Data Fusion* patterns are implemented based on XSLT technology. Finally, we are currently evaluating MicroStrategy as a browsing and analysis tool for mining the Palliative Warehouse.

5. Related work

The idea of component-based software development has become an important part of modern software engineering methods. Component-based software systems are assembled from a number of pre-existing pieces of software called *software components* (plus additional custom-made program code). Software components should be (re)usable in many different application contexts. Rather than being an independent paradigm in its own right, the introduction of *connection-based programming* [Szy98] has been driven mainly by the introduction of component-based software engineering. In this paper, we combine ideas of component-oriented software engineering with our experiences about the architectural requirements of third generation Web IS, in order to develop a catalogue of component patterns that facilitate the construction of such systems.

Our work is related to the notion of architectural styles [SG96]. Architectural styles do not result in a complete architecture but can rather be seen as an architectural framework. Architectural styles are specific views for one subsystem at different level of a system. In contrast, architectural patterns [BMR96] are problem oriented. They express and separate the concerns of fundamental structures in software systems

The *Façade* pattern defines and provides a unified interface to a set of interfaces in a subsystem [GHJV95]. The presented Data Portal pattern is an architectural variant of the Façade design pattern. Data Portals provide unified interfaces to components of the system. A possible technology-driven instantiation of Data Portal is the *Abstract Database Interface* pattern [ABM96]. Abstract Database Interface makes an application independent from the underlying database platform.

Buschmann at al. [BMR+96] define two architectural patterns that are related to Data Connection. The first pattern is called *Broker* and coordinates communication of decoupled components that interact by remote services invocations. Second, *Pipes and Filters* provides filter components which encapsulate processing steps for data streams. The pipes pass through the data between adjacent filters. Further, Gomaa, Menascé and Shin [GMS01] described component interconnection patterns for synchronous, asynchronous and brokered communication. A designer of new distributed application can use their patterns for appropriate component interaction. In contrast to those communication centric patterns, Data Connection mediates data and can be compared to the *Mediator* pattern [GHJV95] at architectural level.

6. Conclusions and Future Work

The World-Wide Web has evolved from predominantly being a hypertext system to being a global network of dynamically interacting information services. Architectures of third generation Web information systems are increasingly complex and integrate all types of devices, ranging from large server back-ends down to embedded household appliances and wearable computers. A key challenge in realising such systems is the management of their evolution: architectures of modern distributed Web IS are constantly in flux and organisations have to be able to respond quickly to changing requirements. We believe that the paradigm of connection-based programming combined with the idea of recurring architectural patterns can be used to facilitate development and evolution of third generation Web IS. The patterns discussed in this paper can be identified in many modern Web IS. However, they often interleave and the separation of concerns is not quite as clear as described in this paper. Making these patterns explicit helps us to engineer and evolve third generation Web IS in a more systematic and efficient way. Preliminary results of our Palliative Care case study indicate the validity of this assumption. Our current and future work is on a more thorough evaluation of this thesis and on exploring the limitations of our pattern catalogue.

Acknowledgements

This research has been supported in part by the National Science and Engineering Research Council of Canada (NSERC) and the Advanced Systems Institute of British Columbia (ASI). We thank Francis Lau and Craig Kuziemsky from the UVic School of Health Information Science for many fruitful discussions on the Palliative Care project.

References

[ABM96] A. Aarsten, D. Brugali and G. Menga. *Patterns for Three-Tier Client/Server Applications*. In Pattern Languages of Programs (PloP96), Monticello, Illinois, 1996.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern Oriented Software Architecture*. John Wiley & Sons, Inc.1996.

[GHJV95] E.Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Pattern: Elements of Reusable Object Oriented Software.* Addison-Wesley, Reading, MA.1995.

[GMS01] H. Gomaa, D. A. Menascé, M. E. Shin. *Reusable component interconnection patterns for distributed software architectures*. In Proceedings of the Symposium on Software Reusability20, Toronto, Ontario, Canada. ACM Press. May 2001.

[Kuz02] C. Kuziemsky *2002 Palliative Care Project Report*. Internal Report - University of Victoria School of Health Information Science. March 2002

[OHE99] R. Orfali, D. Harkey, J. Edwards. *Client/Server Survival Guide*. Third Edition. Wiley. 1999.

[SG96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an emerging Discipline*. Prentice Hall, 1996.

[Szy98] C. Szyperski. *Component Software Beyond Object-Oriented Programming*. Boston, MA, Addison-Wesley and ACM Press. 1998.

[XLE] http://www.alphaworks.ibm.com/tech/xle.