ELSEVIER

Data & Knowledge Engineering 22 (1997) 261–281

DATA & KNOWLEDGE ENGINEERING

Modeling application domains

Mohammad Ketabchi[a,*], Kamyar Jambor-Sadeghi[1,b]

[a]*Computer Engineering Department, School of Engineering, Santa Clara University, Santa Clara, CA 95053, USA*
[b]*Technology Deployment International, 4100 Moorpark Ave., Suite 116, San Jose, CA 95117, USA*

Received 11 September 1995; revised 12 April 1996; accepted 31 October 1996

Abstract

Modeling the interactions among the objects in an application domain and the activities in the domain are necessary extensions of modeling the structure and behaviour of objects in the domain. Object models capture the structure and behaviour of components in the application domains. Interaction models describe how the components in the domain interact to realize the activities in the domain. Activity models are used to describe the functionality of the domain. The integration of object, interaction and activity models is essential for producing *complete domain models* which we refer to as Component–Interaction–Activity (CIA) models. The CIA model of a domain simplifies the development of applications in that domain because it makes extensive reuse of the elements of the domain possible. Concepts for developing the CIA models of application domains are described. A Domain Modeling Language (DML) which provides constructs for expressing the concepts in CIA models is introduced. The DML provides: (i) a class construct to define components, (ii) rule, trigger, event and relationship constructs to define interactions among components, and (iii) process and task constructs to define the activities in the domain. A Domain Modeling Tool which provides a graphic interface to DML and makes the CIA domain models expressed in DML executable is introduced. An example of using CIA models in developing applications is presented.

Keywords: Object-orientation; Application domain modeling; Application development

1. Introduction

An application domain model consists of *objects* of the domain, *interactions* among the objects of the domain, and *activities* of the domain. Objects are the building blocks that interact with each other. The interactions among objects are of various types: some objects are related to other objects through various associations, some objects react to certain operations in other objects by performing operations of their own, and some objects react to other objects when those other objects reach certain states of interest. The activities in the domain are processes that define the sequences of tasks that need to be performed to achieve certain domain functions.

*Corresponding author. Email: ketabchi@otl.scu.edu
[1] Email: kjambor@tdiinc.com

PII S0169-023X(96)00056-0

The notions of object [1], composition [2], aggregation [3], generalization/specialization [4,5], classification [6] and relationship [7] can be used to describe the structure of the domain under consideration. The notion of object operation can be used to describe the protocols of objects where the protocol of each object defines the behaviour of the object [8]. However, information about the interactions among objects and the activities in the domain would not be captured. There should be mechanisms to capture such information because the model of interactions and activities of an application domain is as useful as the model of objects in the domain. Moreover, making the model of interactions and activities explicit would make the objects more generic, easily reusable and maintainable.

A complete domain model would integrate interaction and activity modeling as well as object modeling. The advantages of such an integrated domain model, which we refer to as Component–Interaction–Activity (CIA) model are:

1. Leads to a better and more complete understanding of the application domains.
2. Allows domain experts to verify the structure and protocol of the objects in the domain by simulating key activities of the domain.
3. Facilitates the design and development of application systems.
4. Simplifies the maintenance of application software because software modifications generally occur due to changes in interactions and activities in the domain. Modeling interactions and activities explicitly rather than embedding them in objects simplifies their modifications.
5. Leads to more reusable domain objects, because unless the interactions and activities are modeled explicitly they tend to be embedded in the object models. This leads to large and complex objects that are hard to reuse.

Section 2 gives an overview of related work. Section 3 presents concepts for developing CIA domain models and shows that objects become manageable and more useful once these concepts are available. Section 4 presents a domain modeling language called DML that provides facilities to formally express CIA domain models. A complete example of building an application using the concepts and facilities introduced in Section 3 Section 4 is given in Section 5. Section 6 provides a summary and describes our approach to the evaluation and validation of the research results presented in this work.

2. Related work

There exists a considerable amount of work in the areas of data modeling, Object-Oriented (OO) modeling and component-based application development. The work related to data modeling has mostly been reported in database literature. The work related to object-oriented modeling has mostly been reported in object-oriented analysis and design literature. The work related to component-based application development has mostly been reported in object-oriented software engineering literature.

Data modeling has been the subject of intense research since the beginning of DBMS technology [9]. [10] gives a comprehensive review of the work on data modeling in the 1970s. The work on data modeling in the late 1970s and early 1980s was focused on extending relational [11], ER [12] and semantic data models [13]. Although many of these models, such as [14], introduced features that are supported by OO models, complete OO data models [15] were not introduced until the mid 1980s. OO

data models can be classified into two groups: OO extensions of relational [16], ER [17] and semantic [18] data models, and database extensions of OO programming languages [19]. The former emphasize the query capabilities and the later emphasize computational completeness. Pure OO models do not include facilities such as rule and trigger.

[20] provides a review of the work on supporting rules, triggers [21,22] and events [23] in DBMS. Although rules, triggers and events which can be used to model interactions have been incorporated in several languages [24], modeling the activities of application domains has not received much attention. Design and specification languages such as SDL (Specification and Design Language) [25] and process modeling [26] and specification languages such as PML [27] focus primarily on processes without relating them to domain structure. Consequently, the binding of processes to structural and behavioral abstractions is generally very weak. The work reported in this article draws from these data modeling experiences and uses the modeling concepts object, relationship, rule, trigger, event, process and task evolved over many years to introduce the three-layer CIA domain model which supports object, interaction and activity modeling to simplify application development. The explicit modeling of object, interaction and activity allows the procedural elements that provide computational capabilities to be contained in the lowest layer (object layer) as methods of generic reusable objects, and declarative elements which provide logic and higher-level control flow to be defined in the upper two layers. This in turn allows new applications to be built using the existing objects and the existing applications to be customized without low-level procedural coding.

There are several OO analysis and design methodologies [28]. An example of this is the Object Modeling Technique [29] which includes object diagrams, data-flow diagrams and state-transition diagrams. These diagrams represent object models, functional models and dynamic models of systems. However, these models are not well integrated. It is difficult to recognize how an object, operation and attribute in the object model relates to a data flow process in the functional model or to an event or state in the dynamic model. Methodologies such as Shlaer and Mellor [30] and Booch [31] emphasize modeling of the structure and behaviour of objects. The thrust of these methodologies has been in identifying the objects needed to build an application rather than developing domain models which can then be used to develop multiple applications. These methodologies are useful in identifying the objects of interest and some of their attributes and operations, and important relationships such as inheritance and composition relationships among objects. We have reviewed several of these methodologies in an attempt to utilize them for the analysis phase of the development of CIA domain models. We have concluded that it is difficult to identify the interaction and activity layers using these methodologies adequately unless they are extended and enhanced.

In recent years, object technology has given rise to a set of powerful component-based software development systems that are referred to as application builders. Object-oriented application builders attempt to raise the level of abstractions in the application development process from programming to refinement and composition of predefined components. To achieve this, current application builders generally provide a set of predefined objects with generic base functionality, and facilities to realize application logics, define new objects and develop user interfaces for the applications.

Examples of object-oriented application builders are NeXTStep [32], Intelligent Pad [33], VisualAge [34], ObjectVision [35] and ObjectBuilder [36]. The current object-oriented application builders generally provide technology domain objects such as graphic user interface objects, database management system interface objects, and generic objects such as tables and dictionaries. They do not provide high-level modeling facilities for domain experts to define application domains. Moreover, the

composition facilities of the current application builders are limited and require programming expertise rather than application domain knowledge only. The work presented in this article allows application developers to build their applications using domain-specific objects, semantics and primitives. Domain experts develop application domain models and application developers build applications by composing the objects in domain models. Application domain models make extensive reuse of domain objects and semantics possible and facilitate integrating various applications within the same domain, because they share objects through the domain model.

3. Concepts for developing CIA models of application domains

The CIA model of an application domain consists of three components: objects with structure and behaviour, interactions among the objects, and activities in the domain. In the pure ER model, the structural components are modeled as Entities and the associations among entities are modeled as Relationships. Entities have no associated behaviour and there exists no modeling constructs to express interactions among the entities and activities in the domain. As a consequence, the gap between the entities and the domain functions is large. Applications have to implement the behaviour of the objects, interactions among them and the activities in the domain. This is illustrated in Fig. 1a.

The Object-Oriented (OO) paradigm introduced the 'object' as a construct that combines structure and behaviour. The structure of the object is described by its properties and the behaviour of the object is described by its operations. The 'data' now is encapsulated with the access methods and, as shown in Fig. 1b, the gap between the entities and the domain functions is considerably smaller. The introduction of the object concept has reduced the gap because the operations that are needed to

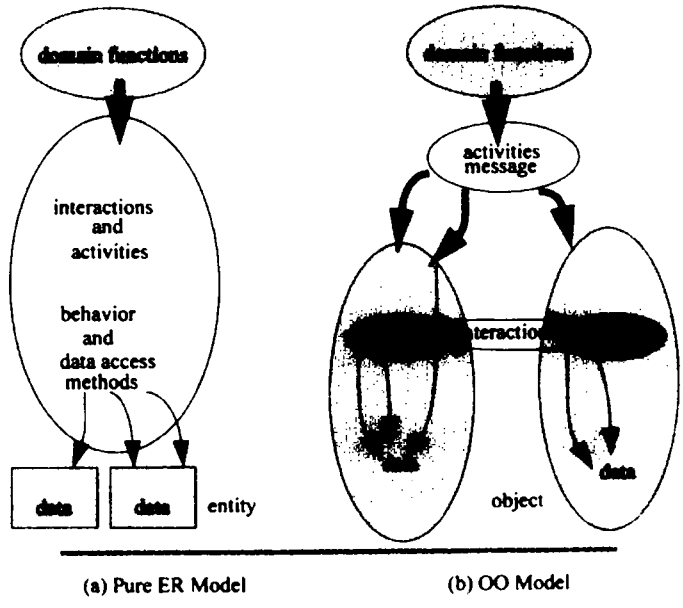
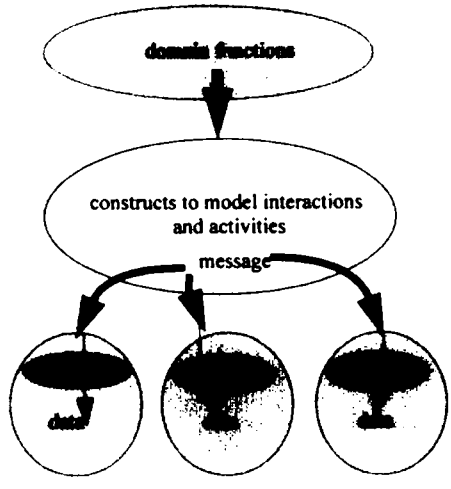


Fig. 1. Evolution of modeling from ER to OO.

access the data are now encapsulated with the object. However, in the current state of OO modeling, there are no concepts that facilitate the modeling of interactions and activities. Consequently, the objects end up encapsulating all the interactions and a significant portion of activities in addition to the operations. This leads to a dissolution of the interaction and activity models, and large complex objects that are hard to reuse and maintain. There is a need to provide constructs that would facilitate modeling of interactions and activities. As illustrated in Fig. 2, a consequence of this would be that objects would diminish in size and the inter-object communication would be minimized. This would, in turn, lead to more reusable and maintainable objects.

An example illustrating the current state of object modeling is shown in Fig. 3. The Student object upon receiving a *remove(course)* message would remove the course from its list of courses. To maintain consistency, the Course object would be sent a message *remove(student)* to remove the student from its list of registered students. Also, the Course object would in turn send a message *refund(amount)* to the Account object so that an appropriate update would be made to the student's account.

Thus there is *no* clear distinction between what describes the object vis a vis what describes the use of the object. This leads to ambiguous models which in turn reduce the extensibility and reusability of the objects in the models. Fig. 4 illustrates the proposed alternative that would allow objects to remain autonomous while capturing the interactions and activities in the domain using constructs other than objects. The Student Withdrawal activity consists of two steps: Drop Course and Refund Fees. These steps can invoke the methods defined on objects. The Drop Course step may invoke the *remove(course)* method defined on the Student object. Similarly, the Refund Fees step may invoke the method *refund(amount)* defined on the Account object. The execution of the *remove(course)* method of the Student may trigger the execution of the *remove(student)* method of Course object. This interaction is defined outside the Student and Course objects by the deleteStudent trigger. The deleteStudent trigger ensures that we have no student in the list of students in a course without having the course in the list of courses of the student. Note that the *remove(course)* method in Fig. 4 is



Extended OO Model

Fig. 2. Desired state of object modeling.

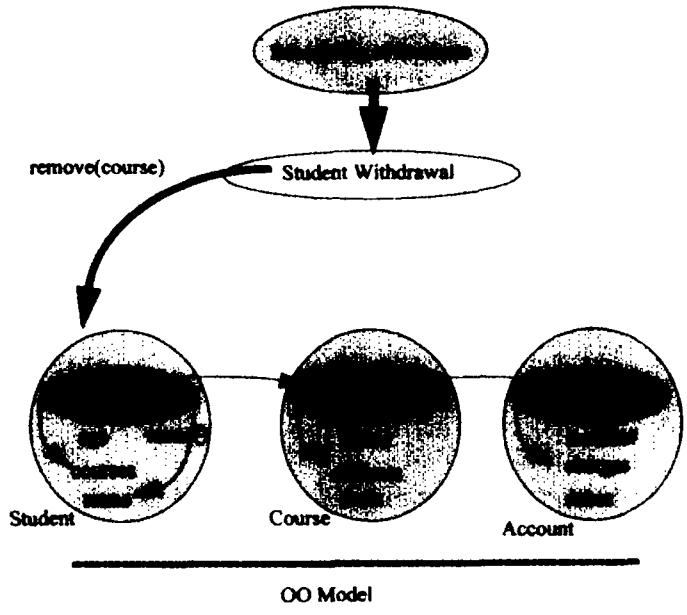


Fig. 3. Current OO design example.

different to the *remove(course)* in Fig. 3 because it does not contain an invocation of the *remove(student)* method. Similarly the *remove(student)* method in Fig. 4 is different to the *remove(student)* method in Fig. 3 because it does not contain an invocation of *refund(amount).*

Relationship, rule, trigger and event are used to facilitate modeling of interactions among objects. A

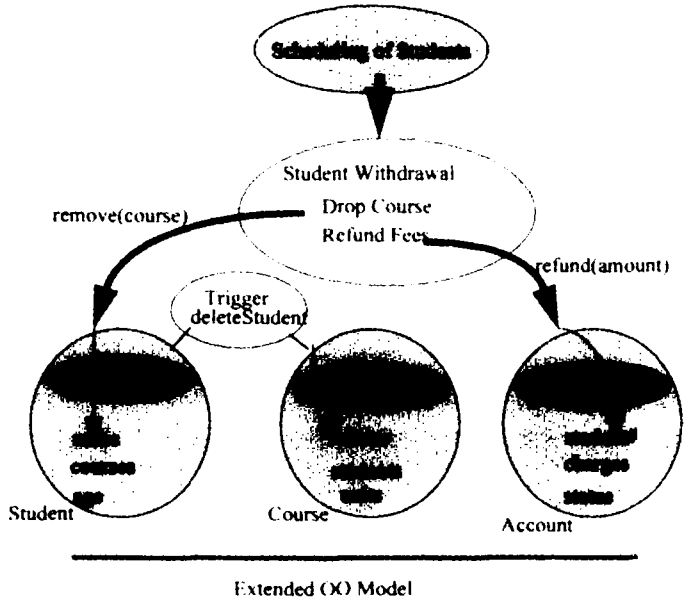


Fig. 4. Desired OO design example.

relationship establishes an association among objects without embedding information about one object inside other objects. If a relationship construct is not available, associations among objects would need to be captured by embedding links in objects. Such links would make it difficult to use one of the objects without using the others. The relationship construct supports certain integrity constraints such as cardinality and facilitates the enforcement of other integrity constraints such as referential integrity [37–39] over participating objects. Moreover, it provides associative access paths from one participating object to others.

In any domain there are certain rules that need to be enforced. When a rule is violated, corrective actions may need to be taken. The rule construct, therefore, should specify the invalid states of the domain and the corrective actions that should be taken when such states occur. A rule may span multiple objects. As an example, we may require that a student can not take a course unless he/she has taken certain prerequisite courses.

A trigger allows domain models to establish an action-based (rather than condition-based as in the case of rule) interaction among objects by interconnecting the behaviour of the objects. The interaction specified by a trigger requires an operation of an object to be executed after an operation of a certain other object is executed. In the example of Fig. 4 the trigger deleteStudent will cause the student information of a Course object to be updated when the course information of the Student object is updated due to deletion of a course. This is achieved through the trigger without modifying any existing behaviour of Student and Course objects and without requiring any knowledge of Course object by Student object.

An event allows domain modelers to establish a type of interaction among objects that occur frequently, but is difficult to express without proper modeling constructs. An event is generated when an object reaches a state which is of interest to some other objects. The course becoming unavailable is the event which may be of interest to other objects such as the Supervisor object which would handle the event by taking appropriate actions. The event construct would allow the Supervisor object to register interest in the Course Unavailable event and specify its way of handling this event.

Process and task are two concepts that would facilitate modeling of activities. A process is an activity integral to the complete description of the application domain. Examples of processes are the 'scheduling' activity of the school domain, the 'check-in, check-out' activities of the hotel domain and the 'patient registration' activity of the hospital domain. Processes can be viewed at different levels of abstraction where each process may be decomposed into other processes which in turn can be further decomposed into smaller and simpler processes until its decomposition is of no interest from the activity modeling point of view.

Processes model activities and therefore represent the functionality of the domain. When a functionality of the domain is invoked, a process is started. To achieve the desired results, methods of objects need to be executed. However, in many cases, methods of objects are low level and do not readily correspond to the steps in the process. Moreover, often several methods need to be invoked to perform a step in the process. The notion of a task is introduced to fill the gap between the meaningful steps in processes and the low-level methods of objects.

A task is an indivisible unit of work which would execute in its entirety, or not execute at all. A task can be viewed as a logical transaction that invokes one or more methods of objects to perform a meaningful step in one or more activities of the domain. Tasks, therefore, tie processes to the objects.

The OO model extended with relationship, rule, trigger, event, process and task leads to a better design because most modifications in the functionality of applications would not require reprogram-

ming objects but would only require modifications of interactions and activities. Moreover, an object can be used independently of other objects because no knowledge of any other objects is built into it.

4. Domain modeling language DML

DML is an object-oriented language which supports the CIA model of application domains by providing language constructs to express the concepts described in Section 3. Fig. 5 shows DML constructs which support the specification of the three layers of CIA domain models.

The logical meta-model of DML is an extended object model which is not limited to classes and the manipulation of their instances. Relationships, events, rules, triggers, tasks, processes and their instances are first-class citizens in DML. In addition to facilities such as encapsulation, classification, generalization, specialization, state, hyperobject and relationships, DML supports dynamic modeling. Asynchronous communication among objects is modeled by DML event construct. Constraints in a domain are modeled by DML rule and trigger constructs. Activities in a domain are modeled using DML process and task. DML provides support for commonly used primitives such as repeater, selector and iterator which eliminate the need for loops and reduce procedural code in manipulating large sets of data. The key DML feature which makes the specification of interactions and activities possible is its set of powerful modeling facilities that allows control flows implementing the common application logics to be explicitly specified rather than embedded in code. The DML has been designed to provide a complete set of facilities to define CIA domain models. The modeling facilities are such that they can be defined on existing objects without requiring them to change and they lend themselves to high-level graphic representation and manipulation.

Since DML class construct is similar to the class construct in object-oriented programming languages such as C + + [40] and Smalltalk [41], and object-oriented database languages such as ODL [19] and IDL [42], its description is omitted. The following sections describe DML relationship, rule, trigger, event, process and task constructs. A more detailed discussion of these and other DML facilities are given in the DML Users' Guide [43].

4.1. Relationship

The DML relationship construct allows domain modelers to establish associations among existing classes of objects without modifying those classes. Note that this facility is in addition to relationships such as the specialization (inheritance) relationship which is defined by subclassing and the composition relationship which is defined by object references. Once the association has been established, operations on the instances of one participant class can be propagated to the related instances of other participant classes. For instance, a trigger defined on the Enrollment relationship

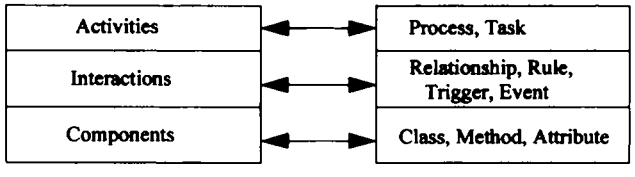


Fig. 5. Constructs for defining CIA domain models.

can print the names of student participants when the operation ShowCourseInfo is executed on the Course participant of the relationship. Furthermore, because the relationship provides an access path from one participant to another, integrity constraints can be defined and enforced among the related instances, and given an instance of one of the participant classes, the related instances of other participant classes can be retrieved. Achieving this functionality in object-oriented languages which do not support relationships requires considerable programming effort.

The DML relationship has structure and behaviour sections. There are three parts in the structure paragraph of a relationship: Participant, RelationshipAttributes and InstanceAttributes. The Participant contains two or more participant specifications. The RelationshipAttributes is similar to the class attributes of the class construct. The InstanceAttributes specifies the attributes associated with the individual occurrences of a relationship and generally model intersection data of the relationship. The behaviour section consists of RelationshipOperations and InstanceOperations. These are similar to the class operations and the instance operations section in the class construct in DML and other object-oriented languages such as Smalltalk. Every DML relationship has a set of translator-generated operations which are used for creating and deleting relationships, and associative accesses to participants. These operations are inherited from the system-defined kernel Relationship.

The DML definition of a relationship called Enrollment is given below. Enrollment has Student, Course and Instructor as its participants, and Date (registration date) as its instance attribute. The cardinality constraint (*m..n*) on a participant specifies that each instance of the participant may participate in at least *m* and at most *n* instances of the relationship. For example, the cardinality constraint (0..50) on participant Course means that each Course object can appear in at most 50 instances of the Enrollment relationship. The multiplicity constraint (*m..n*) on a participant indicates that each instance of the relationship may have at least *m* and at most *n* instances of the participant. For example, the multiplicity constraint (1..1) on participant Student means that an instance of the Enrollment relationship must have exactly one Student object.

BeginRelationshipSpec
RelationshipName Enrollment;
SuperRelationshipName Relationship;
Structure
Participants
s: Student Multiplicity(1..1) Cardinality(1..50);
c: Course Cardinality(0..50);
i: Instructor;
InstanceAttributes
date: Date;
EndRelationshipSpec

The cardinality and multiplicity constraints are optional. The default cardinality is (0..*n*) and the default multiplicity is (1..1).

4.2. Rule

The DML rule construct allows modelers to specify invalid states of the domain model, and to provide corrective actions which will be executed if an invalid state occurs. DML rules are defined on

the existing classes in the model without requiring them to be modified. This is in contrast to those systems which define rules as part of objects. DML rules are declarative and do not violate the encapsulation of objects.

A DML rule has Context, When, OnRequest and Maintain sections. The Context section declares the type of the variables used in the other sections of the rule. The When section describes the condition under which the rule must be checked. The Condition specified in the Maintain section must be true after the execution of the method in the OnRequest section, otherwise the rule is being violated. When a rule is violated, the action specified in the IfViolated section of the rule is executed. This is consistent with the Event–Condition–Action (ECA) model of rules deployed for an object system where the When and OnRequest sections are similar to Event, Maintain is similar to Condition, and ifViolated is similar to Action in ECA rules. If no condition is specified in the When section and no method is specified in the OnRequest section, the rule must hold true always. The following shows an example of rule which states that a part-time student cannot register for more than three (3) courses.

BeginRuleSpec
RuleName maxClassCount;
Context c:Class;
st:Student;
When st.parttime();
OnRequest st.addCourse(c);
Maintain st.getClassCount() < = 3;
IfViolated error ('Part-time student should not register for more than 3 courses!');
EndRuleSpec

4.3. Trigger

The DML trigger augments the behaviour of domain objects by adding additional semantics without changing their existing operations. However, instead of being a constraint, as in the case of rule, trigger interconnects the behaviour of objects.

A DML trigger has Context, When, OnRequest and Perform sections. The Context section is similar to that in the rule construct. The When section specifies a state or a condition. The OnRequest section specifies a method of an object or a task. After the invocation of the method or task specified in OnRequest, the method specified in the Perform section is executed only if the condition specified in the When section holds. If the When section is not specified, the meaning of always is implied. The following shows an example of DML trigger. This trigger specifies that after a student adds a course, the student count for that course should be updated.

BeginTriggerSpec
TriggerName updateStudentCount;
Context c:Class;
st:Student;
When

OnRequest st.addCourse(c);
Perform c.updateStudentCount();
EndTriggerSpec

4.4. Event

The DML event construct is a mechanism to specify a pattern of communication among objects. The communication pattern includes an object expressing interest in certain changes in the state of other objects, and wanting to be notified, or specifying actions that must take place when the changes happen. The change in the object state which is of interest is specified in the event definition. The manner of handling an event is specified at the time of registering interest in the event. The same event may elicit different responses in different object instances.

All DML events inherit methods, register(), notify() and check() from the system-defined DML Event. A DML event has Context, RaiseAfter, When and Operations sections. An event has associated internal data structures which store the values of the variables in the Context section when the event is raised. Additionally, when an object registers interest on an event, it can pass arguments to the register operation. The values of these arguments at the time when the event is raised will also be stored. The RaiseAfter section specifies an operation after which the condition specified in the When section should be checked. In case no operation is specified, a meaning of always is implied. The When section specifies the state which resulted because of the changes the object has undergone. The Operations section allows the methods in the System Event to be overwritten. The following shows an example event, Promotion, which is defined over Professor class and is raised when an employee is promoted.

BeginEventSpec
EventName Promotion;
Context p: Professor;
RaiseAfter promote();
When p.title > p.oldTitle;
Operations
EndEventSpec

Both trigger and event provide mechanisms for control propagation. However, trigger specifies which objects are to respond and how they should respond, while event only specifies what to respond to. The objects which respond and the manner in which they respond are decided at the time of registering interest on the event.

4.5. Task

The DML task construct is an abstraction of a basic unit of work in a domain which does not clearly belong to a single object class. Defining a task as a method of a single object class will, therefore, create logical inconsistency and unnecessary dependencies among the classes involved. A task is performed in its entirety or not at all.

A DML task has Parameter, Context and Steps sections. The Parameter section allows the task to

accept parameters from the invoker which may be a process or another task. The Context section specifies the declarations of local variables used in this task. The Steps section contains a sequence of statements to be executed when the task is invoked. The following shows an example task called AssignAdvisor. This task accepts a student and a department. It selects a professor who is not on sabbatical and has the least number of advisees in the department and assigns him/her as the advisor of the student.

```
BeginTaskSpec
TaskName                    AssignAdvisor;
Parameter
                            s: Student;
                            d: Department;
Context
                            L: [Professor];
                            p: Professor;
Steps
                            L = selectAvailableProfs(d);
                            sortByAdviseeCount(L);
                            p = L.getFirst();
                            s.set-advisor(p);
EndTaskSpec
```

A task is executed when it is invoked from within a process. The process which invokes a task provides the execution environment for the task.

4.6. Process

The DML process construct represents an abstraction of a collection of steps that realize a functionality of the domain. A DML process can be directly invoked. For example, student admission, student registration and class scheduling are candidates for processes in the University domain. A DML process has HyperObjects, Context and Steps sections.

The HyperObjects section specifies one or more HyperObjects which provide the name scope for the process. A HyperObject is an abstract object representing a collection of DML constructs which together as a whole is semantically significant. A HyperObject may contain Classes, Relationships, Events, Rules and Triggers. HyperObjects provide an abstraction mechanism for organizing domain knowledge into larger units of information than individual modeling construct. By including a HyperObject in a process specification, all member constructs of the HyperObject are accessible to the process. The Parameter section allows the process to accept parameters from the invoker which may be another process. The Context section specifies the declarations of local variables used in this Process. The Steps section contains a sequence of statements to be executed when the process is invoked.

A DML process is independently executable. It may be invoked directly by the users or by other processes. Each process is executed concurrently with other processes in the same application. A process has its own execution environment and a thread of control starting from the first statement in

the Steps section. The environment of each process contains all the constructs such as classes, relationships, rules and events that are needed for the process to be executed successfully. Any tasks or methods invoked directly or indirectly by the process have access to the process environment. In the current version of DML, a DML process is implemented by a UNIX process.

An example of process is the student admission process in the university domain. The process admits a new student to a department and assigns a new student id and an advisor to the student. The DML definition of the process is shown below.

BeginProcessSpec
ProcessName Admission;
HyperObjects University;
Context
d: Department;
dname: String;
s: Student;
name: String;
phone: Phone;
gpa: Real;
Steps
InputDept(dname);
d = Department.locate(dname);
InputStudent(name, phone, gpa);
s = Student(name, phone, gpa);
s.set-id(generateNewId());
AssignAdvisor(s, d);
EndProcessSpec

In addition to class, relationship, rule, trigger, event, process and task constructs, DML provides selector, iterator and repeater constructs. The selector construct allows a subset of objects that satisfy a given condition to be selected from a larger set of objects. The iterator construct separates the selection of objects in a set from their processing. Every time the iterator defined on a set is invoked, it returns one element of the set based on a selection criterion defined as part of the iterator until there are no more elements. The repeater construct takes a set of objects and an operation and applies the operation to each element of the set.

4.7. Domain modeling tool DMT

DMT [44] consists of a graphic user interface to DML, a DML to C + + translator and a C + + encapsulator. The graphic user interface of DMT supports the creation of application domain models using DML constructs. Through the simple 'point-and-click' actions of the mouse, iconic forms of the DML constructs are created and their various properties such as name, inheritance relationships, context, participants and references are defined. This is similar to sketching the model on paper. The modeler can then select each construct in the sketch and specify its details using forms that are

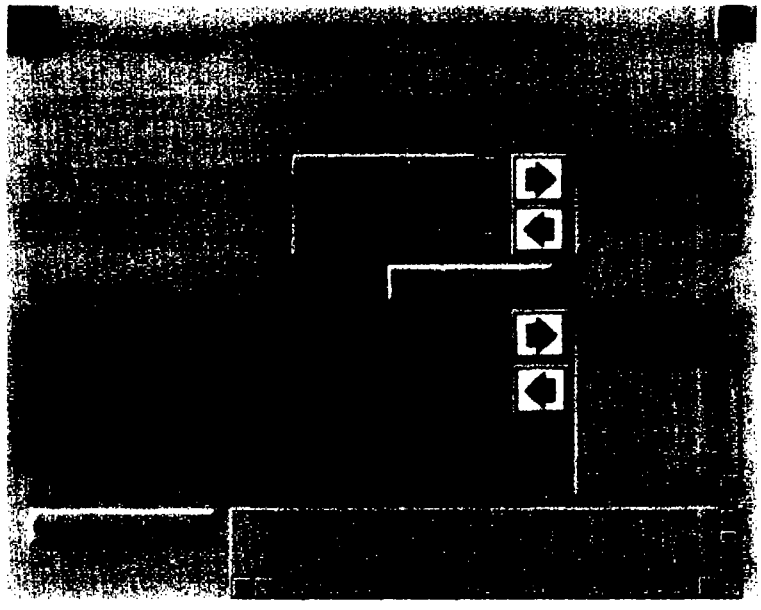


Fig. 6. Definition of the Student class using DMT.

presented by the system and are partially filled by the information already available through the sketch. The specifications of the details and sketching may be interleaved and are iterative.

Fig. 6 depicts a session with DMT. The window displays the DML definition of the Student class. The user can browse/edit the various properties of the Student class.

The models developed using DMT can be saved and edited at a later point in time for extensions and modifications. DMT models may also be defined using DML textual representation and then browsed and edited through the DMT graphic user interface. The DML translator generates C + + code for DML models. The DML encapsulator encapsulates C + + classes in DML classes. Once a class has been encapsulated, DML users can define other DML constructs, such as events and triggers on the class to enhance its functionality. DMT allows textual and graphic representations of domain models and C + + classes to be translated to one another. A detailed description of DMT implementation is given in [45].

In addition to providing a convenient way for defining models using DML, on-line help and error checking, DMT provides extensive browsing, filtering and abstraction facilities to simplify the construction and use of large domain models. Any construct in the model can be searched for by name, any types of constructs that are not of interest at the time of browsing the model can be filtered out, and any types of constructs that are the focus of attention can be expanded for more detailed representation.

5. Using application domain models

Application development is still by and large a time-consuming and labor-intensive task and often results in broken deadlines, inflated budgets, buggy products and even project failures. An application

development system, which allows composition of applications using CIA domain models can reduce the time and effort needed for developing applications. Once a CIA domain model is defined, multiple applications can be built using the model with substantially less effort than will be required otherwise, because the model provides a higher-level platform for application developers. The applications built on top of a CIA domain model share the components in the domain and can be integrated through a persistent repository whose global schema is the domain model.

An application built on top of a CIA domain model is an implementation of an activity in the domain together with the graphic user-interface and input–output facilities. To develop applications using a CIA domain model, composition tools are needed, which allow the components generated from the model to be composed to implement activities. An application development system called OCADS (Object-Centered Application Development System) has been developed based on this approach at the Object and Multimedia Technology Laboratory in Santa Clara University. OCADS allows users to quickly build CIA models of an application domain, then construct multiple applications by visually composing the components. OCADS has two major components: DMT, which employs DML as its host language for domain modeling, and a visual composition tool which allows the composition of applications. The details of the implementation of OCADS are beyond the scope of this paper. Interested readers are referred to [45]. In this section we describe the steps for the development of a prototype application for a typical activity in the University domain. Let us consider the registration activity of a part-time student. Assume:

1. A part-time student cannot register for more than 3 courses.
2. All prerequisite courses must have been completed before a student is allowed to register for the course.
3. To register for a course the student must have good credit. Otherwise approval from the Students' accounts must be obtained.
4. A student cannot register for a course which is full. A course is considered full if the number of registered students in the course has reached the maximum student count. A student may register for a course that is full if he/she obtains instructor's approval.
5. A registered student must be added to the list of students taking the course.

Two rules, *MaxCourseCount* and *CompletePrereq* are defined to meet the requirements 1 and 2 stated above. The MaxCourseCount rule states that a part-time student may never register for more than 3 courses. The CompletePrereq rule states that a student can register for a course only if he/she has completed all the pre-requisite courses. The DMT tool used for creating the MaxCourseCount rule is shown in Fig. 7.

Two events, *CreditCheckFailed* and *CourseIsFull* are defined to meet the requirements 3 and 4 stated above. The CreditCheckFailed event is raised when a student is discovered to have a negative balance. Accounting manager will put a stop on the student's registration when this event is raised. The CourseIsFull event is raised when a course gets full. Schedule supervisor, instructor and Dean may be interested in this event. The DMT tool used for creating the classIsFull event is shown in Fig. 8.

The following tasks can be defined for the requirements stated above:

- VerifyCredit: verify that a student is in good financial standing.
- ClearCredit: pay the money owed and make sure the accounts reflect the correct balance.
- RegisterStudent: Register student for the course.

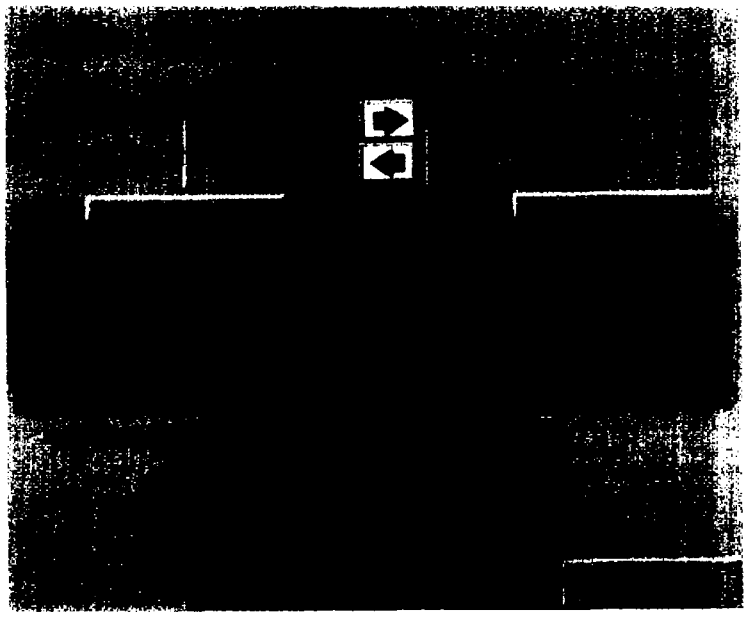


Fig. 7. Definition of the maxCourseCount rule using DMT.

A trigger, *UpdateCourseList* can be defined to meet the requirement 5 stated above. This trigger states that after a student is accepted to a course the course list must be updated.

Having defined all the events, rules, triggers and tasks, the registration activity can be defined as a DML process, as shown in Fig. 8. Note that in an actual situation many of the rules, triggers, events and tasks would have already been defined as part of the domain model.

When the activity *CourseRegistration* is invoked, execution of the process shown in Fig. 9 will be initiated. The steps of the process will be executed sequentially as specified in the process. The

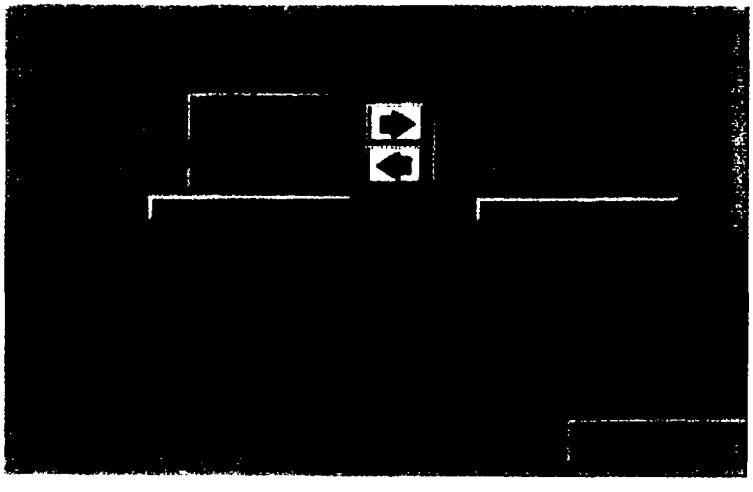


Fig. 8. Definition of the classIsFull event using DMT.

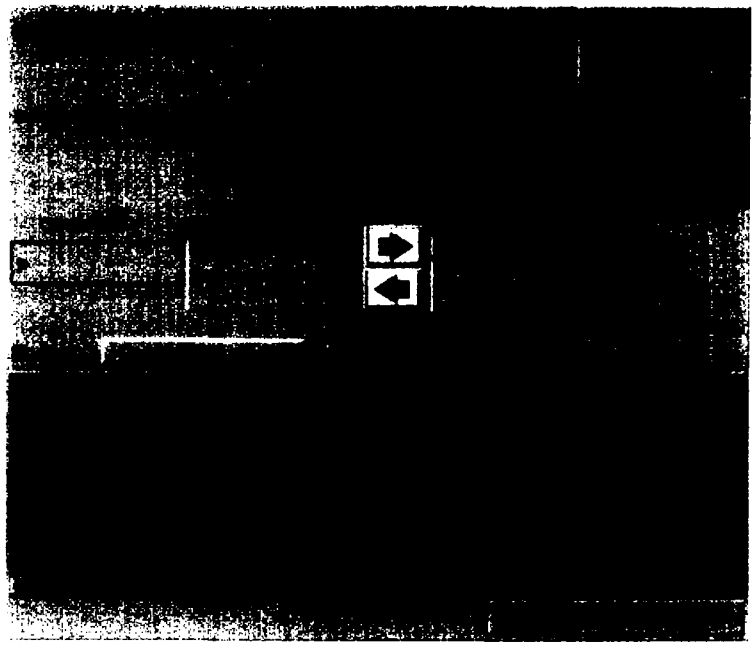


Fig. 9. Definition of the CourseRegistration process using DMT.

execution of a step may invoke a task which in turn may invoke a method. The execution of a method may raise an event or fire a trigger or a rule. Events raised in a process, as well as triggers and rules fired in a process, execute in the context of the process. If an object has registered interest on an event, its handler will be executed immediately after the completion of the operation which causes the event to raise. If an object has requested to be notified when an event is raised then the object has to check the notification when it wishes to respond to the event. Events have precedence over triggers which have precedence over rules. In other words, if an operation causes events to be raised and triggers and rules to be fired, events will be processed first, the triggers will be processed next, and, finally, rules will be processed.

6. Summary and concluding remarks

A three-layer model of application domains referred to as the Component-Interaction-Activity (CIA) model was presented in this article. A CIA model results in objects that are more reusable and simplifies developing applications which can easily be modified and extended. A CIA model consists of a component layer defined using a class construct, an interaction layer defined using relationship, rule, trigger and event constructs, and an activity layer defined using task and process constructs. Class construct is used to specify component types. Relationship construct establishes associations among otherwise independent objects and provides associative access to objects. Rule and trigger establish dynamic interactions among instance objects based on the properties of the objects at type level. Events allow instance objects to register interest in certain changes of states in other objects and

react to such changes. Tasks, which are atomic units of meaningful work in the domain, tie together activities and objects.

A Domain Modeling Language referred to as DML was described which provides facilities to express a CIA model of application domains. The implementation of the first version of DML has been completed at the Object and Multimedia Technology Research Laboratory in Santa Clara University. This prototype includes a graphic user interface called DMT, a DML to C + + code generator, and a C + + to DML encapsulator. DMT provides a high-level graphic and form-based interface for DML users. DML encapsulator encapsulates C + + class libraries as DML classes. The prototype is being used at the Object and Multimedia Technology Research Laboratory in a case study which involves developing Telecommunication Management Network (TMN) domain model and applications prototypes.

From the beginning of the OCADS project in 1990 we have been concerned with the validation and evaluation of its results. We could not take a mathematical approach to this task because mathematics, being an abstraction, hides key qualities such as ease of use and flexibility which are important for an application development system. Although it is possible to develop an 'ease-of use' or 'flexibility' algebra by identifying the key factors which are relevant to these qualities, such algebra is nonexistent at this point and its development was clearly outside the scope of the research project reported in this article. Another approach was to develop a prototype and conduct usability experiments and case studies. Although we have developed a relatively complete and robust prototype system based on the ideas described in this article, comprehensive experimental studies are expensive time-consuming and involve many factors which are hard to control. The third approach was to conduct systematic comparative studies. We chose to take this approach.

We identified a set of application logic primitives which are the building blocks of most sophisticated applications. We then developed these application logics using several existing object-oriented application builders and object-oriented languages C + + and Smalltalk, and our prototype. As a completeness test we verified that all the application logic primitives could be specified using DML. We compared the time and effort needed to develop these applications with each system. We also compared the resulting applications with respect to the size of the code which we had to develop and its complexity.

The set of application logic primitives were defined to exercise the following features and functions. Each primitive application was developed to exercise a single criteria:

(i) Expressive power:
- Classification.
- Generalization.
- Composition.
- Polymorphism.
- Rule and trigger.
- Event and notification.
- Task and process.
- Selector and iterator.

(ii) Reuse:
- Reuse of existing components such as C + + class libraries.
- Reuse of the element of a domain model across applications in that domain.

(iii) Ease of use.

(iv) Ease of modification.

In addition to the primitive applications which exercise individual criteria we developed applications which would exercise the interactions among multiple criteria. This exercise has been very useful and has so far generated interesting results. In the process we have identified the strengths and weaknesses of our approach and the prototype. The key deficiencies of our system are a lack of an analysis methodology to assist the domain experts, and a lack of tools to check the consistency of the CIA domain models. The primary strength of our system has been the extensive reuse of components and the simplicity of development and modification of applications once the CIA domain model is developed. Of course, domain models are never complete. They tend to extend as new applications are built; however, over time, extensions needed for developing new applications decrease and the amount of reuse increases. The future work in this project will be pursued in two directions:

1. Domain analysis methodologies and tools which assist the domain experts in identifying not only the objects and relationships in their domain but also the rules, triggers, events, as well as tasks and processes of the domain, need to be developed. Our approach is to extend and enhance an existing OO analysis and design methodology to cover this need.
2. Algorithms and tools to help the domain experts ensure the consistency of CIA domain models need to be developed. Checking the consistency of the set of rules and the synchronization of processes are well understood but non-trivial tasks. We believe the key issues in checking the consistency of CIA domain models are the synchronization of events and the interplay of rules and triggers with events.

Acknowledgments

The work on DML design and development project has been funded by Fujitsu Japan and Fujitsu Network Transmission Systems. Several researchers at the Object and Multimedia Technology Laboratory at Santa Clara University have contributed to the development of the prototype, among them Rani Mikkilineni and Surapol Dasananda deserve special recognition.

References

[1] S. Khoshafian and R. Abnous, *Object-Orientation: Concepts, Languages, Databases, User Interface* (Wiley, New York, 1990).

[2] M. Ketabchi and V. Berzins, Mathematical model of composite objects and its applications for organizing efficient engineering databases, *IEEE Trans. on Software Engineering* (January 1988).

[3] B. Ehlmann and G. Riccardi, A notation for describing aggregate relationships in an object-oriented data model, *Proc. 1st Intern. Conf. on Applications of Databases*, Vadstena, Sweden (June 1994).

[4] S. Spaccapietra, C. Parent, K. Yetongnon and M.S. Abaidi, in N. Prakash (ed.), *Generalizations: A Formal and Flexible Approach, Management of Data* (Tata McGraw-Hill, 1989) pp. 100–117.

[5] M. Ketabchi, V. Berzins and K. Maly, Generalization per category: Theory and application, *Proc. Intern. Conf. on Information Systems* (1985).

[6] J. Smith and D. Smith, Database abstractions: Aggregation and generalization, *ACM* 2(2) (1977).

[7] A. Albano, G. Ghelli and B. Orsini, A relationship mechanism for a strongly typed object-oriented database programming language, *Proc. 17th Intern. Conf. on Very Large Databases*, Barcelona (1991).
[8] O. Nierstrasz, A survey of object-oriented concepts, in Won Kim and Frederick H. Lochovsky (eds.), *Object-oriented Concepts, Databases and Applications* (Addison-Wesley, 1989) pp. 3–21.
[9] R. Elmasri and S. Navathe, *Fundamentals of Database Systems* (Benjamin Cummings, 1989).
[10] D.C. Tsichritzis and F.H. Lochovsky, *Data Models* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[11] C.J. Date, *An Introduction to Database Systems* (Addison-Wesley, 1986).
[12] P. Chen, The Entity–Relationship model—Toward a unified view of data, *ACM Transaction on Database Systems* I (March 1976) pp. 9–36.
[13] R. Hull and R. King, Semantic database modeling: Survey, applications and research issues, *ACM Computing Surveys* 19(3) (1987) 201–260.
[14] E.F. Codd, Extending the database relational model to capture more meaning, *TODS* (December 1979).
[15] M. Ketabchi, V. Berzins and S. March, An object-oriented semantic data model for CAD applications, *J. of Information Sciences* 46 (September 1988).
[16] *ISO-ANSI Working Draft: Database Language SQL3*, X3H2/94/080 and SOU/003 (1994).
[17] S. Spaccapietra and C. Parent, in Peri Loucopoulos and Roberto Zicari (eds.), *ERC + : an Object Based Entity–Relationship Approach, Conceptual Modeling, Databases and Case: An Integrated View of Information Systems Development* (John Wiley, 1992).
[18] S.Y.W. Su, V. Krishnamurthy and H. Lam, *An Object-Oriented Semantic Association Model (OSAM*)* (American Institute of Industrial Engineers, 1988).
[19] R.G.G. Cattell, *The Object Database Standards: ODMG-93* (Morgan Kaufmann Publisher, 1994).
[20] J. Widom and S. Ceri (eds.), *Active Database Systems—Triggers and Rules for Advanced Database Processing* (Morgan Kaufmann Publishers, 1996).
[21] Y. Caseau, Constraints in object-oriented deductive databases, *Proc. 2nd Intern. Conf. on Deductive and Object-Oriented Databases* (1991).
[22] E. Baralis, S. Ceri and S. Paraboschi, Declarative specification of constraint maintenance, *Proc. Intern. Conf. on the Entity–Relationship Approach*, Manchester, UK (December, 1994).
[23] N. Gehani, H.V. Jagadish and O. Shmueli, Event specification in an active object-oriented database, *Proc. of ACM SIGMOD Intern. Conf. on Management of Data*, San Diego, California (June 1992).
[24] R. Hull and D. Jacobs, Language constructs for programming active databases, *Proc. 17th Intern. Conf. on Very Large Databases*, Barcelona, Spain (September 1991).
[25] F. Belina and D. Hogrefe, The CCITT specification and description language SDL, *Computer Networks and ISDN System* (North Holland, Amsterdam, Vol. 16, 1988–1989) 311–341.
[26] V. Markowitz, Representing processes in the extended E–R model, *Information Sciences* 52 (1990) 247–284.
[27] R.F. Bruynooghe, J.M. Parker and J.S. Rowles, PSS: A system for process enactment, *1st Intern. Conf. on the Software Process* (Oct 1991) pp. 128–141.
[28] D. Monarchi and G. Puhr, A research typology for object-oriented analysis and design, *Commun. ACM* 35(9) (Sep. 1992).
[29] J. Rumbaugh et al., *Object-Oriented Modeling and Design* (Prentice Hall, Feb. 1991).
[30] S. Shlaer and S. Mellor, *Object-Oriented System Analysis: Modeling the World in Data* (Prentice Hall, 1988).
[31] G. Booch, *Object-Oriented Design with Applications* (Benjamin/Cummings, 1991).
[32] S. Karfinkel and M. Mahoney, *NeXTstep Programming Step One: Object-Oriented Application* (Springer-Verlag, 1993).
[33] Fujitsu, *IntelligentPad User's Guide, V1.0* (Fujitsu, Japan, 1994).
[34] *VisualAge Users' Guide and Reference Manual*, 2nd Edition (IBM Corporation, October 1994).
[35] *ObjectVision 2 Reference Guide* (Borland International, 1991).
[36] *ObjectBuilder C + + User Interface Builder User Guide Release 1.0* (ParcPlace Systems).
[37] C.J. Date, Referential integrity, *Proc. 7th Intern. Conf. on Very Large Databses*, Cannes, France (1981).
[38] V.M. Markowitz, Referential integrity revisited: An object-oriented perspective, *Proc. 16th Intern. Conf. on Very Large Databases*, Brisbane, Australia (1990).
[39] Rumbaugh, Controlling propagation of operations using attributes on relations, OOPSLA'88, *ACM SIGPLAN* 23(11) (November 1988).

[40] B. Stroustrup, *The C + + Programming Language* (Addison-Wesley, Reading, MA, 1991).
[41] A. Goldberg and D. Robinson, *Smalltalk-80: The Language and its Implementation* (Addison-Welsey, Reading, MA, 1983).
[42] *The Common Object Request Broker: Architecture and Specification* (Object Management Group, July 1995).
[43] M. Ketabchi, S. Dasananda, R. Mikkilineni and X. Li, *DML User Guide, Technical Report, TR#OTL-1994-8* (Object Technology Lab., Santa Clara University, 1994).
[44] A. Antablian, Z. Li and M.A. Ketabchi, *DMT User Guide, Technical Report, TR#OTL-1994-9* (Object Technology Lab., Santa Clara University, 1994).
[45] R. Mikkilineni and M. Ketabchi, Encapsulation and enrichment—Towards improving the reusability of software components, *Information and Systems Engineering*, in press.
[46] S. Adams, *Meta Methods: The MVC Paradigm* (HOOPLA, July 1988).

Dr. Kamyar Jambor-Sadeghi is a senior software engineer at TDI. His interests include advanced object-oriented database tools. He received a B.S. in Electrical Engineering from the University of the Pacific, and an M.S. in Electrical Engineering and a Ph.D. in Computer Engineering from Santa Clara University.

Mohammad A. Ketabchi received his Ph.D. in Computer and Information Sciences from The University of Minnesota, Minneapolis. He is a Full Professor of Computer Engineering at the Computer Engineering Department of Santa Clara University. He is the director of the Object and Multimedia Technologies Research Laboratory. He is the founder President of Technology Deployment International Inc., an advanced software development corporation in San Jose, California. His research interests include object technologies including OODBMS, Internet/intranets component-based application software architectures and formal languages.