ICIS 1992 Proceedings

International Conference on Information Systems (ICIS)

1992

MODELING REQUIREMENTS FOR FUTURE: ISSUES AND IMPLEMENTATION CONSIDERATIONS

Pentti Marttiin
University of Jyvaskylli

Kalle Lyytinen
University of Jyvaskylli

Matti Rossi
University of Jyvaskylli

Veli-Pekka Tahvanainen
University of Jyvaskylli

Kari Smolander
University of Jyvaskylli

See next page for additional authors

Authors
Pentti Marttiin, Kalle Lyytinen, Matti Rossi, Veli-Pekka Tahvanainen, Kari Smolander, and Juha-Pekka Tolvanen

MODELING REQUIREMENTS FOR FUTURE: ISSUES AND IMPLEMENTATION CONSIDERATIONS

Pentti Marttiin
Kalle Lyytinen
Matti Rossi
Veli-Pekka Tahvanainen
Kari Smolander
Juha-Pekka Tolvanen
Department of Computer Science and Information Systems
University of Jyväskylä

ABSTRACT

In this paper, we discuss some requirements for future CASE (Computer Aided Software/Systems Engineering) environments. These requirements include increased modifiability and flexibility as well as support for task and agent models. We claim that they can only be addressed by developing more powerful representation and modeling techniques. As a possible basis for a modeling technique, we propose the GOPRR (Graph-Object-Property-Relationship-Role) data model, which addresses some of these requirements. In addition, a general information architecture for a future CASE environment is outlined. It includes three kinds of models for methodology specification: meta-datamodels, activity (task) models, and agent models. These models are defined using the GOPRR model with some additional concepts for IS development process and agent participation.

1. INTRODUCTION

Interest in extending the functionality of Computer Aided Software Engineering (CASE) tools has been spurred by the rapid advances in computing power, object-oriented database management techniques, and graphical interfaces. As a result, numerous integrated tool and support environments have been launched during the last years. In general, a CASE tool can be seen as a mechanism that empowers an information system (IS) developer through supporting a variety of development tasks by managing and manipulating different kinds of IS representations. As Forte and Norman (1992) point out, CASE tools have been successful in automating many routine software development tasks. The most common way to support development tasks using current CASE tools is to help in elicitation tasks by deriving graphical IS specifications, and then transforming them into textual representations for correctness checking and reporting. Therefore these tools fail to address several important features in IS development (ISD) and have problems with their views of (meta) data, process and group work support, and technical features.

Most current CASE tools lack customizing capabilities, support only a fixed set of methods, and thereby operate on a fixed metamodel. During the last years, however, new CASE shells[1] with combined textual/graphical/matrix representation support and changeable repository (meta data) have appeared on the market. Thus, possibilities for building a customizable repository are already there. Examples of commercial CASE shells are *VSF* (Pocock 1991) and *Paradigm+*. Other research oriented CASE shells with similar features are *RAMATIC* (Bergsten et al. 1989), *MetaEdit* (Smolander et al. 1991), and *MetaView* with its graphical extension *GI* (Sorenson, Tremblay and McAllister 1991).

The main reason for CASE tools penetration was the need for improving the quality of systems development processes and productivity (Osterweil 1987; Charette 1986). Support for development processes is still lacking in most CASE tools. However, these tools can produce IS descriptions according to a method, but they contain no knowledge of what methods can be used in specific development phases, how the descriptions are produced, what is the output of the phase, and how the outputs should be validated. In organizations where users with different abilities participate in systems development projects and where the decision making needs close co-ordination between managers and projects, it is important to support development processes with output validation mechanisms.

Another lacking feature has been the failure to address systems development as a group activity. Recently, several attempts have been made to broaden the scope of CASE environments[2] to support Computer-Supported Cooperative Work (CSCW), business modeling and reengineering, strategy formulation and architecture specification (Conklin and Begeman 1988; Chen, Nunamaker and Weber 1989; Rose, Maltzahn and Jarke 1992). Clearly, support for these tasks is needed if early phases of systems development will be covered by CASE.

CASE researchers are also searching for solutions for specific technical design issues that are currently under supported in CASE such as transformation support (Brinkkemper et al. 1989; Boloix, Sorenson and Tremblay 1991; Rossi et al. 1992), reengineering (Bjerknes et al. 1991), and version control (Katz 1990; Hahn, Jarke and Rose 1991). Object-orientation (Booch 1991; Rumbaugh et al. 1991) and supplementing CASE environments with hypertext features[3] (Conklin and Begeman 1988; Garg and Scacchi 1990; Cybulski and Reed 1992) have been proposed as solutions to some of these problems.

In general, CASE environments address the obvious need to produce, analyze and manage descriptions of various kinds during interactive system development (ISD) in a consistent, efficient and user-friendly manner. This need can be addressed as *a metamodeling problem*: what are the requirements for languages to describe and model data stored, represented, and manipulated in the IS repository and what is the process to specify the content and functionality of such a repository? We define *metamodeling* to be the task that produces a metamodel for ISD. Accordingly, a *metamodel* embraces a methodology specification.[4] In general, this issue has been largely ignored until recent years and commercial CASE tools suggest only ad hoc solutions to this problem. The goal of this paper is to specify the requirements for the information architecture of the future CASE (shell) environments that will help to address these problems in a more systematic manner.

The paper is organized as follows. In section 2, we take a look at the future and try to characterize an ideal CASE environment by the turn of the millennium. In section 3, we specify the elementary requirements for representing methodology data in such an environment. In section 4, we discuss how to meet these requirements within an integrated environment. Finally, in section 5, we draw some conclusions of the desired level of methodology support in the future CASE environments.

2. "2001: A CASE ODYSSEY"

To give an idea of a future CASE environment, we outline here a scenario of a session using such an environment.

Think of a usual morning in the software development center of NanoSoft Corp. A software engineer and project manager S.E. logs into his/her Beta+ workstation and picks up a "CASE Factory" icon in the "WorkSpace manager." The "CASE Factory" icon starts the CASE environment's desktop window. It shows as icons his/her current projects and the tools s/he has picked for use. S/he then sets up the environment for a certain project, in which a graphical "view manager" provides tools to modify or look at the different parts of the IS specification. S/he begins to work by modifying the contents of a project's schedule. The schedule comes up as an activity net in a graphic window. Being more comfortable with textual definition of tasks, s/he changes the type of the window from graphic to text and continues his/her work with a task list. After having modified the project s/he looks at the electronic white board where one of his/her project members notifies him/her about a problematic OER-design. S/he closes the schedule window and clicks the "OER-diagram" icon, causing the OER-diagram to appear, and locates the problematic part by searching for the note assigned to it. S/he sketches a solution for the problem by backtracking the design into an earlier version and then starts a negotiation session to converse about the solution with other project members. An automatic repository agent also takes part in the discussion by showing those parts of the project that are affected by the design change. When the others have accepted the solution, s/he stores the new design into the repository and continues with his/her other duties. Meanwhile, the repository agent will check and annotate all affected documents and possibly update some transformations and documents automatically.

As a project manager, S.E. uses the same shell with special reporting tools to monitor the state of the project and to estimate the cost and risks of the project. When, for example, s/he needs the FPA++ cost analyzing software, s/he can initiate it from within the environment, because the FPA++ tool conforms with the "common software platform" standard, and the local CASE environment manager has attached it to the CASE repository allowing FPA++ to use the repository services. As S.E. performs these monitoring operations repetitively, s/he has specified semiautomatic and automatic agents that perform routine tasks (e.g., collect managerial data, check the consistency of IS specifications, do library management and notify of slippage from the schedule) for him/her, in a manner similar to the assistants in *The Programmers' Apprentice* (Rich and Waters 1990).

When the users of the CASE environment find a modeling technique inappropriate, the CASE environment manager can use the CASE administrator's toolset (C. Martin 1988) to modify the repository in order to overcome the problems. The toolset gives the CASE environment manager the means to meet the tool users' different needs such as organizational standards and the users' modeling experience. It gives him/her the opportunity to reconfigure tools and the repository to capture the peculiarities of the evolving universe of discourse (the domain of systems development activity).[5] Hence, the repository can be customized during the project without necessarily loosing earlier work. The user (agent) profiles that maintain data of the project members, their positions and abilities, are an integral part of the repository. Using this data, the CASE environment manager can specify access rights and project assignments within the administrator's toolset. Accordingly, the same person can appear in both the roles of a software engineer and a CASE tool administrator if his/her user profile allows this.

These requirements for the next generation environment are to some extent known and accepted as, for example, C. Martin (1988) and McClure (1989) show. Their implementation, however, remains a major research challenge. When these demands are met, those users who instead of CASE tools prefer word processors, graphical packages, and compilers will start to fully utilize "real" CASE environments.

3. REQUIREMENTS FOR METAMODELING IN CASE ENVIRONMENTS

To build next generation CASE environments such as the one outlined above, we need to understand the necessary properties and functions of such an environment. In this section, we present a method representation framework that forms an essential functional component of the future CASE environments. The principle here is to first define the simplest requirements for data representation and then to derive the generic principles that are needed if a CASE environment is to fully support all the requirements that are typical for data storage, transfer, and representation during IS development. Accordingly, we have divided the modeling requirements into the following three interrelated levels of complexity:

1. requirements for representing IS data related to the use of a single method,

2. requirements for representing IS data related to the use of multiple connected methods, and

3. requirements related to describing the IS development process and IS developer groups.

To cope with the complexity of the higher levels, the tool developer needs to understand the mechanisms and principles of the lower levels. Therefore, we claim that CASE environment with a full life cycle coverage must be able to represent different methods, to interlink these into chains or support packages (methodologies), and finally to handle the users' or groups' requirements such as the work style and interactions.

3.1 Single Method Representation

We distinguish between the following levels of representation for a single method. The representations can be implemented in a textual, tabular, or a graphical form (cf. Lyytinen, Smolander and Tahvanainen 1989).

- *Two-dimensional representations*

  The two-dimensional representations define the modeling principles to describe the relationships between atomic or complex data items. These are typical requirements for specifying methods such as Data Flow Diagrams (DFD) (Gane and Sarson 1979) or Entity-Relationship (ER) modeling (Chen 1976) into a CASE environment. The modeling principles needed here are classification/instantiation, generalization/specification, and cartesian and cover aggregation.

  Classification/instantiation is presented in metamodeling in terms of type-instance pairs. Generalization/specification is handled with **is_a** and **is_a_kind_of** relationships, for example a *flow from process* in a DFD diagram is_a_kind of *flow* (cf. Smolander et al. 1991). The cartesian aggregation defines that an object is constructed from its components (attributes) and the cover aggregation specifies that a complex object consists of other objects. An example of the former is *"A car has attributes model name, model year, color, and miles driven"* and of the latter is *"A car consists of parts: door, wheels, and motor."*

- *Operational semantics defined over these representations*

  These include the mechanisms for specifying semantically correct update semantics, operation granularity, transactions, object identification, and naming. The update semantics specify the sequence of operations for adding, changing, and deleting the contents of an IS repository that will preserve the object semantics. Operation granularity defines the level on which the operation arguments are specified, for example update locks can be set either on the model or on the object level. In the object-oriented terminology, the operations correspond to methods defined for each modeling concept class and the notion of transaction is defined by a chain of methods covering a set of classes.

- *Three or n-dimensional representations*

  This involves modeling and implementing recursive data structures such as decompositions used in data flow diagrams. A detailed discussion of decomposition principles is given by Wand and Weber (1989), where several decomposition types are introduced and a general decomposition model is developed. A general model of n-dimensional representations and the associated consistency rules is needed to handle recursively organized diagrams in a semantically correct way. A possible solution for the graphical representation of decompositions is presented by Harel (1988).

- *Representations of domain knowledge*

  Domain knowledge relates to the use of a method. It is required for validating representations with regard to business models, application area, architectures, and method knowledge. Principles of supporting business models within a CASE environment are discussed in more detail by Chen, Nunamaker and Weber (1989). These include modeling concepts that guide managers' thinking and activities (such as organizational structures and goals, business processes and strategies, strategic assumptions, critical success factors) and the relationships between them. As such method knowledge grows, one can also build knowledge-based tutorials and let the methods "mature" through evolutionary steps and versions encompassing different levels of complexity.

3.2 Multiple Connected Methods

Here we are dealing with issues that arise when several methods are interwoven with one another in IS development, for example, during different development phases or by applying alternative points of view for the same design or implementation problem. Three aspects of this problem can be distinguished here.

- *Horizontal consistency*

  This can be defined as the consistency between different method descriptions on the same level of abstraction. For example DFD and ER models must be cross-checked for the data definition in case the DFD stores are also modelled as ER entities. Lehman and Turski (1987) call this the *verification of a model.*

- *Vertical consistency*

  Lehman and Turski (1987) call this the validation of a model. Vertical consistency implies the maintenance of semantically equivalent descriptions on different levels of abstraction and through the phases of the ISD life cycle. For example, an ER model must be translated into a semantically equivalent relational database schema. We must also bear in mind the problems of reengineering, in which we essentially want to translate a description or a piece of code to a description on a higher level of abstraction and take a look at the representations produced in the earlier phases of the development life cycle.

- *Dynamic consistency*

  Dynamic consistency means the capability of keeping a consistent trace of model changes. In more common terms, this is usually referred to as the version control problem, which has been addressed in several version control systems such as SCCS and RCCS in UNIXTM. Katz (1990) has described general requirements for version modeling and representing different modeling strategies.

3.3 User Related Requirements

CASE tools are used by multiple users who all set varying requirements on the functionality of the CASE tool. Therefore, CASE tools are expected to describe and maintain knowledge of their use environments and also instantiate possible scenarios of their use. A tentative list of the needed user related requirements follows.

- *User role models and the associated subschemes for the IS repository*

  User roles can be divided into roles in the host organization and in the different projects. The first ones build up to an organization model of the larger ISD environment. The user roles in projects define potential use scenarios of the CASE environments; i.e., they permit users to accomplish various ISD tasks. For example, they can permit users to delete specific models or to participate in voting on designs as members of the project management team.

- *Communication models and specifications of the usage of the IS repository in user-to-user communications*

  Communication models are closely linked to role models. They support computer assisted communications, which provide mail services (group mail) and coordinate group work in different tasks of ISD (for example, voting/ranking mechanisms in requirements analysis and review cycles in the design phase or release and configuration mechanisms in maintenance).

- *Method usage knowledge: selection guidelines for different situations*

  Method usage aids explain where to use different methods and different representations. These aids help to define and store formal (or informal) representations of the methodology phases, specifications of their applications, and general method handbooks for phases, and to maintain a history of the method uses and their success rates.

- *Explanations of method use connections*

  These explanations are based on the ordering of the phases and tasks in a methodology. These give guidance to the developers for using the methodology in a "correct" way. These explanations describe what are the specified horizontal, vertical, and dynamic interconnections between the methods in the meta-datamodel. Otherwise, misunderstandings can arise, leading, for example, from one unnecessary transformation model to another although these models need only a horizontal connection.

- *Justifications for making specific choices in design situations*

  Justifications to make design choices are based on the availability of decisions made during earlier designs and on the knowledge of the qualities of the chosen models. This can be done by collecting histories of design trajectories and of the design steps followed (i.e., keeping track of the tasks and the decisions made) in every phase and by organizing and storing the arguments in favor and against the possible models (see e.g., Hahn, Jarke and Rose 1991).

- *Quality criteria for models*

  This helps the validation and verification of the IS models. Validation can be defined as a strive to achieve a best match between the IS models and the "reality." Generally this is a very difficult issue. Some proposed ways to achieve a better match are higher levels of participation, enhanced group processes, simulation, and the use of prototypes and mock-ups. Verification mechanisms can be implemented using the semantics of the data model, verification constraints, and other specified rules.

Some of these requirements have been taken into account in some CASE tools. The most common requirement is the possibility to specify an access control mechanism for the user roles. Some CASE tools even include review cycle mechanisms (Hahn, Jarke and Rose 1991).

4. HOW TO SUPPORT THE METAMODELING REQUIREMENTS WITH AN INTEGRATED INFORMATION ARCHITECTURE

In this section, we propose an architecture to support the metamodeling requirements discussed in section 3. We present the underlying principles of an information architecture that allows one to model the data (IS models generated), the behavior of the information system development (ISD), and the usage of the derived meta data (user related requirements).

4.1 Information Architecture

Our aim is to model different methods and methodologies adequately with as small a set of concepts as possible. Another objective is to outline an information architecture that can be successfully used in modeling methodologies and implemented into a CASE environment to support computer-aided methodology engineering (CAME). We first describe the information architecture, the contents of models on different architecture and levels. We then demonstrate how these models can support the metamodeling requirements stated above.

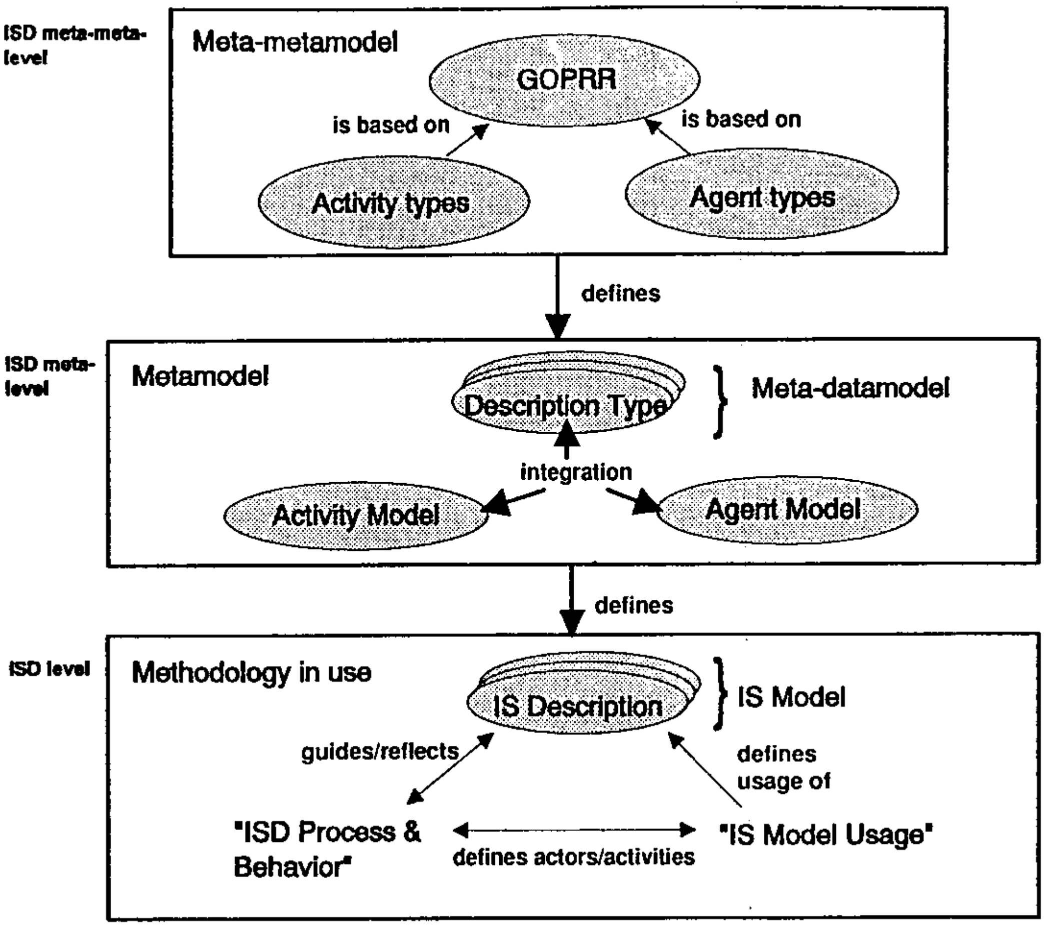

Figure 1. The Information Architecture

Figure 1 shows the three general levels of the information architecture: *ISD meta-metalevel, ISD metalevel* and *ISD level*. First we take a look at the methodology used in IS development. The IS model in the ISD level represents an existing IS or a future system at specific time in IS development. It contains a set of IS descriptions (e.g., "accounting" DFD-diagram or "customer-product" matrix). In general, support for the methodology covers more than the specified syntax for IS descriptions. It needs to contain guidance for creating and manipulating process descriptions, i.e., how and in what order the users can and should work with the models. Because there are several parties involved in IS development, we also need to organize their work (i.e., specify the actors and activities) and define the use modes (e.g., access rights, transactions) for IS descriptions. If we want to define such methodology support, we need a metamodel containing the following three ISD metalevel models.

- The *meta-datamodel* contains one or more interrelated *description types*. We define a description type to be a model of one method or technique. Examples of such description types are "specifications" of *DFD* and *ER* models. A metamodel also contains the mechanisms to specify the connections and transformations discussed in section 3.2.

- The *activity model* supports the follow-up of a methodology application (method usage), for example, by directing the users through the development phases or suggesting steps for a prototyping process.

- The *agent model* defines the access to and use of the IS models during the development tasks and encapsulates the operations (such as querying), controls (access rights, access control), and coordination tasks (such as mail support) available for the users in different roles.

Because the objective is to support different methods and methodologies, we also need a higher level model (meta-metamodel) for defining meta-datamodels, activity models, and agent models, as well as mechanisms to integrate them.

The concepts used in the meta-metamodel are classified in terms of *GOPRR-primitives, activity types,* and *agent types*. The GOPRR-primitives are used to define the *meta-datamodel*, the model that specifies the static part of the methods. These primitives form the core of the offered functionality of the CASE environment and they are described in more detail below.

We first describe what the GOPRR-primitives are and why they are chosen for modeling the meta data in the IS environment. We have carried out extensive metamodeling tasks using the OPRR (Object-Property-Role-Relationship) data model[6] and demonstrated its capability to support "flat" models (without decompositions or complex objects) and in particular, its capability to deal with graphical representations (see Smolander 1991b, Smolander et al. 1991). However, this level of functionality is not sufficient to support all of the requirements listed in section 3.1.

In order to better support all of the requirements, we suggest some extensions to the OPRR model. We call the resulting model GOPRR (Smolander 1991a). Its fundamental primitives are a graph (**G**), an object (**O**), a property (**P**), a relationship (**R**), and a role (**R**). A graph is a collection of the other primitives chunked into some larger meaningful unit: it "collects" these primitives, binds the objects with their roles to participate in relationships, and links the respective representational features (symbols) to the primitives. An *object* is a thing that exists on its own.[7] A *relationship* connects objects in some semantically meaningful way through *roles*, which specify how the objects participate in the relationship. A *property* is an attribute to an object, a relationship, a role, or a graph. Graphical representations of these primitives and their connections are shown in Figure 2.

A major strength in using the GOPRR model in building the information architecture is its close affinity with graphical representations. In Table 1, we depict two dimensions: the *conceptual-representational* dimension and the *type-instance* dimension (Smolander et al. 1991) in modeling and locate our primitives in these dimensions. For each type (primitive), we have a corresponding representation definition, which specifies the graphical[8] behavior, i.e., the available options for drawing an instance of this type. A view definition can override these and specify the instances to be represented in the graph type with specific representational features. A *data type* specifies that property belongs to integer, real, string, text, collection, link, time, or some other domain. The *link* property creates an association between objects or properties. It is used to create the horizontal connections we described in section 3.2. We add a "creation" time-stamp to every primitive when it is created. Using the various *time* properties, we can add temporal time semantics to the primitives as in (Theodoulidis, Loucopoulos and Wangler 1991).

At the instance level, an object instance is represented by an object symbol, a relationship instance by a connector line between object symbols, and a role instance by a line-end symbol (e.g., different arrowheads). Property values are stored in data fields which are shown as labels. Finally, the graph is represented by a group of interconnected graphical representations of the other primitives, for example, as an ER-diagram.

Figure 3 shows the GOPRR specification of GOPRR itself.[9] That is, GOPRR is used here both as the graphical modeling language and the method modeled. The GOPRR model is a graph that can contain graphs, non-properties (the "technical ancestor primitive" for object, roles and relationships), and properties. Objects can participate in many roles and each role must contain at least one object. A relationship has two or more roles denoting binary or n-ary relationships, respectively. Objects, roles and relationships are generalized to form the class of non-properties. Any of these can be exploded to a graph, which can include a set of objects, relationships, and their bindings through their roles. The content and behavior of the non-properties and graphs are described using properties. Ob-

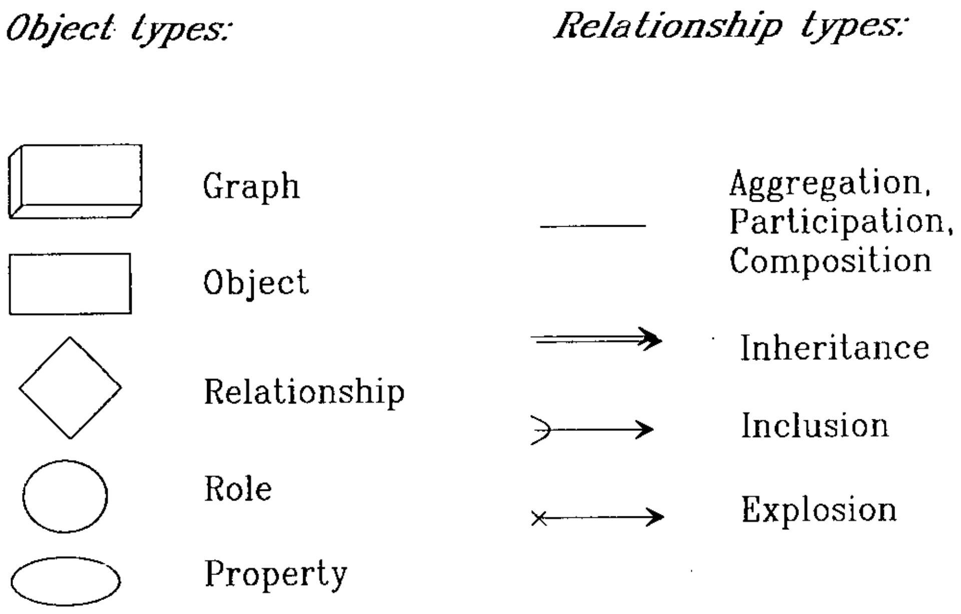

Figure 2. Graphical Representations of Object Types (Primitives) and Relation Types of GOPRR

Table 1. Levels of GOPRR

ISD meta-metalevel	Representation definition	ISD metalevel	Representation
Graph Type	View definition	Collection of instances	Collection of representations
Object type	Symbol definition	Object instance	Object symbol
Property type	Data type	Property value	Data field
Relationship type	Line type	Relationship instance	Connector
Role type	Symbol definition	Role instance	Terminal symbol

jects in GOPRR have a single inheritance, which means that an object inherits the properties of and participates in the same roles as its ancestor object.

Using GOPRR, we can define several integrity checks. For example, we can define the minimum inputs and outputs for objects such as a "process" using roles. Also, with different relationships, we can define what kind of flows can be connected to a "data store." However, in order to increase the quality of checking complex descriptions, we have to extend the GOPRR model with rules. The rule primitive serves several different purposes in CASE environments (e.g., transition, constraint, and verification rules; see also the classification to static and dynamic rules in Persson and Wangler 1990). Here we classify the rules used in defining the meta-metamodel into *basic rules* (those constraints that cannot be defined using the basic GOPRR primitives alone), *prototype rules* (usually design guidelines, which permit users to violate them), and *transformation rules* (describing how one description type is transformed to another). Both the basic and prototype rules can be active (checked automatically) or passive (checked on an explicit trigger). The transformation rules are passive and triggered when a transformation takes place. In the GOPRR model, the rules can be attached to any primitive. This means that in a DFD graph type we can attach rules, for example, to the diagrams, to the processes or the flows of the diagram, or to the properties of the flow. The implementation of rules in CASE environments has been discussed by Chen (1988), Boloix, Sorenson and Tremblay (1991), and Rossi et al. (1992).

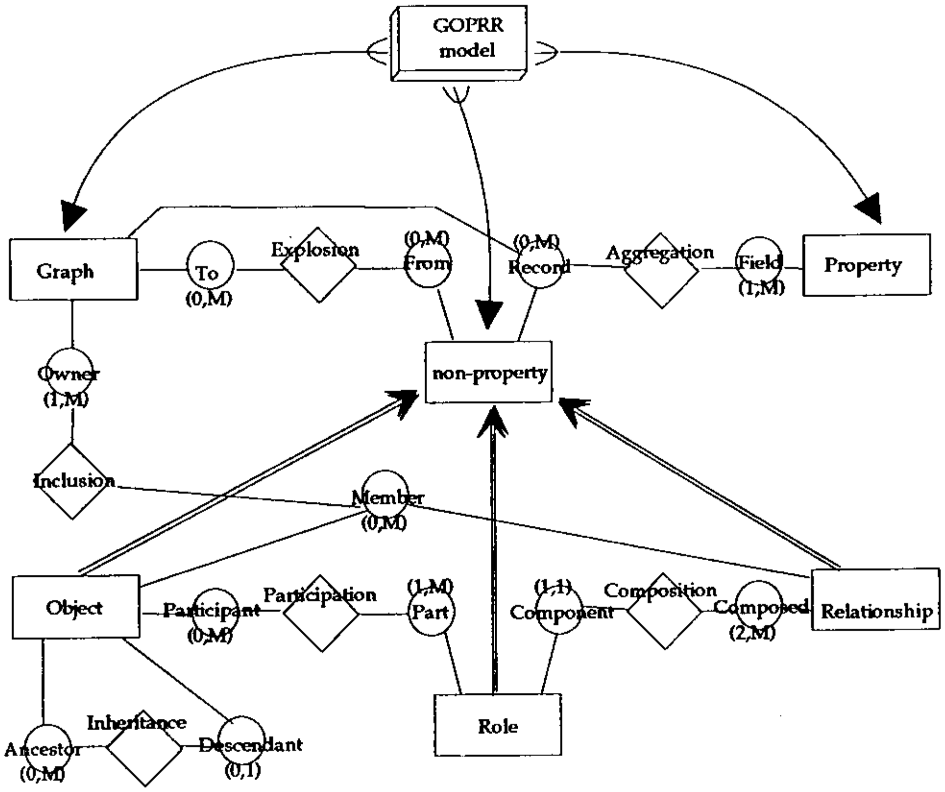

Figure 3. The GOPRR Representation of GOPRR

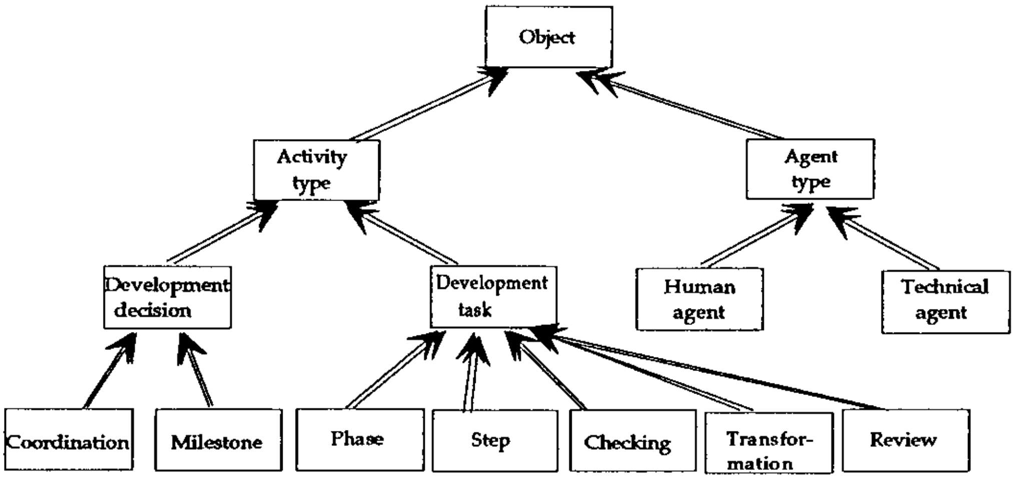

Figure 4. The Specializations of Activity and Agent Types

The activity model describes the users' behavior when following the methodology. The purpose of the activity model is to help, support and co-ordinate the developers' work and to help in maintaining a history of the development tasks and decisions. Similar dynamic models for CASE environments have been introduced by Verhoef, ter Hofstede and Wijers (1991) and Heym and Österle (1992). The activity types used in the activity model can be divided into *development tasks* including development phases, steps, transformations, integrity checkings and reviews, and *development decisions*, for example, situations that occur when one has to choose one of the several development alternatives available (see Figure 4). We have divided decisions to informal coordination decisions and formal milestones where outputs have to be confirmed. Activity types as specialized objects inherit the object semantics described in the GOPRR model, so they can also participate in other types of relationships than those listed in Figure 1. For example, one activity type can precede another.

An agent type can furthermore be specialized to a *human agent* (to organize humans or other "collections of responsibilities" into different roles such as "system analysts" and "project"). Human agents can deploy (perform or trigger) *technical agents*, which are simple procedures or operations such as sending a group mail to all project members. Agent types can be seen as specialized objects which are related to IS model objects and activity types.

We cannot say exactly what are the activity types and agent types needed in different organizations. Our aim here is to find the most general and useful types. The object inheritance in GOPRR allows one to specialize further various activity and agent types (the hierarchy in Figure 4 can be the basis for more detailed types).

4.2 How the Information Architecture Can Meet the Modeling Requirements

4.2.1 Support for single Method Representation

The first group of metamodeling requirements presented above contains support for two-dimensional, three-dimensional, operational semantics, and domain knowledge representation. The two-dimensional representations can be, by and large, modeled with the information architecture. To support classification/instantiation, we have made a distinction between types and instances in Table 1. Other abstraction mechanisms are implemented by the following fixed associations included in GOPRR.

- Generalization/specialization is handled with the inheritance mechanism for objects. We can make specialized objects such as activity types and agent types, which can participate in more specific relationships and contain more properties.
- Decomposition is modeled with the relationships *inclusion* and *explosion* shown in Figure 2.
- For the cartesian aggregation, we use the aggregation between graph and property or between non-property and property. To enable the cover aggregation (complex objects), we can use the same mechanism as for decomposition.

The operational semantics for IS methods are derived from the semantics of the meta-metamodel. GOPRR suggests rules for updates: each update has associations from each primitive to other primitives. A simple example is deleting a relationship instance (on the IS level), where all of the following primitives have to be removed: the relationship, all of the roles and properties of the relationship, all properties of the roles, and the links to the objects involved in the relationship. We divide the semantics further between the concepts and their representations. So, in our example, all representations of the removed concepts have to be removed. At the same time, we must check all of the rules associated with the deleted primitives. The operation granularity, transactions, object identification, and naming mechanisms depend on the selected repository mechanism and information architecture.

The last item in this group is domain knowledge. We can build business models (business rules, functional or business models) graphically and then define the human agents to perform the business tasks. The tasks of the business model definition are described in the process model. Using the metamodeling approach, the redesign and enhancement of the models should be possible and allow evolution of models as the business evolves.

4.2.2 Support for Multiple Connected Methods

A meta-datamodel is composed of one or more description types. Horizontal consistency implies language completeness, i.e., "correct" intersections between description types. There are several mechanisms which can be used to enhance and model these intersections. First, we can use the same object in different description types (specified in GOPRR as graph types). In a view definition, the graph type can specify a different representation for a object, and thereby exclude properties, change relationships, or a changed representation. Another way is to create links between identified objects with a specific *link* property. The latter is used, for example, when we have to describe that an object necessarily follows another.

Vertical consistency implies transformations between description types on different levels. To achieve this, we have to specify transformation rules as a part of the meta-datamodel. Transformation rules can be added to the methodology specification when creating a transformation task in the activity model. The activity models can also be used to define the usage patterns that hold between interconnected methods. Finally, the agent model can specify what parts of a method description a specific user in a project group can modify. To handle dynamic consistency, the time-stamping mechanism, and the capability to exclude specific properties through view definitions can be applied.

4.2.3 Support of User Related Requirements

We can build an agent model to take care of the first two issues of the user related requirements: the organization-wide or project-directed role models and communication models. Organizations can build role models to organize resources and allocate them to projects. Projects can build life cycle models that help to co-operate during IS development. The communication model describes the desired communication patterns in a project or an organization.

To support method usage, we can use knowledge of the meta-datamodel and the activity model and the proposed guidelines for modeling. In the first case, we develop procedures to check the relations between the descriptions and to monitor in which ISD phases the description types are used. In the second case, a prototype rule is transmitted to the developer as a guideline message when s/he is attempting to do something incorrect. The prototype and transformation rules are also used in explaining the method connections.

For justifying the choices, we can collect a history of decision situations as a part of the activity model and thereby maintain a version history mechanism (due to time stamping and versioning of objects) and allow the developers to build software tools to browse this knowledge base for finding the reasons for specific design decisions.

Verification of a model can be handled with the semantics of GOPRR, horizontal consistency mechanisms, and verification rules (rules for checking completeness and protot;pe rules for enforcing the models to retain specific properties). The methodology engineer is responsible for developing the validation rules for a new method or a methodology, although the environment provides some architectural support such as rule building aids, reusable rule and report models, simulation and fast mock ups. For validating documents and other outputs, we can specify particular review tasks into an activity model.

5. CONCLUSIONS AND FUTURE WORK

We described the general requirements for the next generation CASE environment, discussed a set of design issues that need be considered when developing a general architecture to support these requirements, and examined how the future environments can meet them. The implications of this paper are twofold. The first one is recognizing the importance of incorporating the conceptual modeling approach to all aspects of CASE environments. This is introduced as support for data, process, and group-work features of methodologies. We have presented how conceptual modeling primitives can be used to define the models belonging to the methodology (meta-datamodel), behaviors and processes associated with the methodology (activity model), and usage of the methodology (agent model). The second implication is the coverage of the aspects needed in the environment. For example, we plan to add facilities to support group work and an automatic transformation of models between different abstraction levels during the ISD.

The future CASE environment, in our opinion, can be described as an evolving organizational knowledge base (design information system) rather than of a passive data store for system descriptions. This implies that future environments must have a set of tools to handle the elicitation of ISD specifications and to guide the users in gathering information on the IS as well as tools to coordinate the development processes. The environment should also offer a seamless integration of the development steps and different types of tools. Finally, the environment should offer users enough flexibility so that when they demand changes, the environment can easily accommodate these changes.

We have several plans for future research. First, we need to examine the coverage and significance of the proposed architecture by defining several methodologies with it (Tolvanen, Marttiin and Smolander 1992). One of our main activities is to design and build parts of the proposed environment and examine its impacts on ISD work. Examining groupware solutions to develop more sophisticated agent models also seems to be a very promising research area.

6. REFERENCES

Bergsten, P.; Bubenko, J.; Dahl, R.; Gustafsson, M.; and Johansson, L.-Å. "RAMATIC — A CASE Shell for Implementation of Specific CASE Tools." TEMPORA T6.1 Report, SISU, Stockholm, 1989.

Bjerknes, G.; Bratteteig, T.; Braa, K.; Kautz, K.; Kaasbøll, J.; and Øgrim, L. "Project FIRE: Functional Integration through REdesign." In O. Forsgren (ed.), *Proceedings of the Fourteenth IRIS (Information Systems Research Seminar in Scandinavia)*, Institute of Information Processing, University of Umeå, Sweden, 1991, pp. 353-362.

Boloix, G.; Sorenson, P.; and Tremblay, J. "On Transformations Using a Metasystem Approach to Software Development." Technical Report, Department of Computing Science, University of Alberta, Canada, 1991.

Booch, G. *Object Oriented Design with Applications.* Menlo Park, California: The Benjamin/Cummings Publishing Company, 1991.

Brinkkemper, S.; de Lange, M.; Looman, R.; and van der Steen, F. "On the Derivation of Method Companionship by Meta-Modelling." In J. Jenkins (Editor), *CASE '89, Third International Workshop on Computer-Aided Software Engineering*. Imperial College, London, United Kingdom, July, 17-21, 1989, pp. 226-286.

Bubenko, J. "Selecting a Strategy for Computer-Aided Software Engineering (CASE)." SYSLAB Report Number 59, University of Stockholm, June 1988.

Charette, R. N. *Software Engineering Environments: Concepts and Technology.* New-York: McGraw-Hill, 1986.

Chen, M. *The Integration of Organization and Information Systems Modeling: A Metasystem Approach to the Generation of Group Decision Support Systems and Computer-Aided Software Engineering.* Unpublished Ph.D. Dissertation, University of Arizona, 1988.

Chen, M.; Nunamaker, J. F., Jr.; and Weber, E. "The Use of Integrated Organization and Information Systems Models in Building and Delivering Business Application Systems," *IEEE Transactions on Knowledge and Data Engineering*, Volume 1, September 1989, pp. 406-409.

Chen, P. "The Entity-Relationship Model — Toward a Unified View of Data," *ACM Transactions on Database Systems*, Volume 1, March 1976, pp. 9-36.

Conklin, J., and Begeman, M. "gIBIS: A Hypertext Tool for Exploratory Police Discussion," *ACM Transactions on Office Information Systems*, Volume 6, October 1988, pp. 303-331.

Cybulski, J. L., and Reed, K. "A Hypertext Based Software Engineering Environment," *IEEE Software*, Volume 9, March 1992, pp. 62-68.

Forte, G., and Norman, R. J. "A Self-Assessment by the Software Engineering Community," *Communications of the ACM*, Volume 35, April 1992, pp. 28-32.

Gane, C., and Sarson, T. *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

Garg, P. K., and Scacchi, W. "A Hypertext System to Manage Software Life-Cycle Documents," *IEEE Software*, Volume 7, May 1990, pp. 90-98.

Hahn, U.; Jarke, M.; and Rose, T. "Teamwork Support in a Knowledge-Based Information Systems Environment," *IEEE Transactions on Software Engineering*, Volume 17, May 1991, pp. 467-481.

Harel, D. "On Visual Formalisms," *Communications of the ACM*, Volume 31, May 1988, pp. 514-530.

Heym, M., and Österle, H. "A Reference Model for Information Systems Development." In K. E. Kendall, K. Lyytinen, J. I. DeGross (Editors), *The Impact of Computer Supported Technologies on Information Systems Development*. Amsterdam: North-Holland, 1992, pp. 215-240.

Information Manager User's Guide. LBMS, Pre-Release Version 1.01, 1991.

Katz, R. "Toward a Unified Framework for Version Modeling in Engineering Databases," *ACM Computing Surveys*, Volume 22, December 1990, pp. 375-408.

Lehman, M., and Turski, W. "Essential Properties of IPSEs," *ACM SIGSOFT Software Engineering Notes*, Volume 12, January 1987, pp. 52-56.

Lundeberg, M.; Goldkuhl, G.; and Nilsson, A. *Information Systems Development — A Systematic Approach*. Englewood Cliffs, New Jersey: Prentice-Hall, 1980.

Lyytinen, K.; Smolander, K.; and Tahvanainen, V.-P. "Modelling CASE Environments in Systems Work." In *CASE89 Conference Papers*, Kista, Sweden, 1989.

Martin, C. "Second-Generation CASE Tools: A Challenge to Vendors," *IEEE Software*, March 1988, pp. 46-49.

Martin, J. *Information Engineering*, Volumes 1, 2, and 3. Lancashire, England: Savant Research Studies, 1988.

McClure, C. *CASE is Software Automation*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

Osterweil, L. "Software Processes are Software Too." In *Proceeding of the Ninth International Conference on Software Engineering*. Washington DC: IEEE Computer Society Press, 1987, pp. 2-13.

Persson, U., and Wangler, B. "A Specification of Requirements for an Advanced Information Systems Development Tool." In S. Brinkkemper and G. Wijers (Editors), *Proceedings of the Workshop on the Next Generation of CASE Tools*. SERC, Netherlands, 1990.

Pocock, J. N. "VSF and Its Relationship to Open Systems and Standard Repositories." In A. Endres and H. Weber (Editors), *Software Development Environments and CASE Technology*. Berlin, Germany: Springer-Verlag, 1991, pp. 53-68.

Rich, C., and Waters, R. *The Programmer's Apprentice*. Reading, Massachusetts: ACM Press, 1990.

Rose, T.; Maltzahn, C.; and Jarke, M. "Integrating Object and Agent Worlds." In P. Loucopoulos (Editor) *CAiSE '92 Advanced Information Systems Engineering*. Berlin, Germany: Springer-Verlag, 1992, pp. 17-32.

Rossi, M.; Gustafsson, M.; Smolander, K.; Johansson, L.-Å.; and Lyytinen, K. "Metamodeling Editor as a Front End Tool for a CASE Shell." In P. Loucopoulos (Editor), *CAiSE '92 Advanced Information Systems Engineering*. Berlin, Germany: Springer-Verlag, 1992, pp. 546-567.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; and Lorensen, W. *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

Smolander, K.; Lyytinen, K.; Tahvanainen, V.-P.; and Marttiin P. "MetaEdit — A Flexible Graphical Environment for Methodology Modelling." In R. Andersen, J. Bubenko, and A. Sølvberg (Editors), *Advanced Information Systems Engineering*. Berlin, Germany: Springer-Verlag, 1991, pp. 168-193.

Smolander, K. "GOPRR — A Proposal for a Metamodeling Method." Internal working paper, Department of Computer Science and Information Systems, University of Jyväskylä, Finland, 1991a.

Smolander, K. "OPRR — A Model for Methodology Modeling." In Veli-Pekka Tahvanainen and Kalle Lyytinen (Editors), *Proceedings of the Second Workshop on the Next Generation of CASE Tools.* Department of Computer Science and Information Systems, University of Jyväskylä, Finland, 1991b, pp. 135-151.

Sorenson, P.; Tremblay, J.; and McAllister, A. "The EARA/GI Model for Software Specification Environments," Technical Report, Department of Computing Science, University of Alberta, June 1991.

Theodoulidis, C.; Loucopoulos, P.; and Wangler, B. "A Conceptual Modelling Formalism for Temporal Database Applications," *Information Systems,* Volume 16, April 1991, pp. 401-416.

Tolvanen, J.-P.; Marttiin, P.; and Smolander, K. "An Integrated Model for Information Systems Modeling." In preparation, 1992.

Verhoef, T. F.; ter Hofstede A. H. M.; and Wijers, G. M. "Structuring Modelling Knowledge for CASE Shells." In R. Andersen, J. Bubenko, A. Sølvberg (Editors), *Advanced Information Systems Engineering.* Berlin, Germany: Springer-Verlag, 1991, pp. 502-524.

Wand, Y.; Weber, R. "A Model of Systems Decomposition." In J. I. DeGross, J. C. Henderson, and B. R. Konsynski (Editors), *Proceedings of the Tenth International Conference on Information Systems.* Boston, Massachusetts, 1989, pp. 41-51.

Welke, R. "Metabase — A Platform for the Next Generation of Meta Systems Products." In *Proceedings of CASE Studies 1988, Meta Systems,* Ann Arbor, 1988.

ENDNOTES

[1] Bubenko (1988) defines a CASE shell as "a tool to define a method or a chain of methods."

[2] A CASE environment supports several methods in several phases of the IS development, whereas a CASE tool need not do so (Lyytinen, Smolander and Tahvanainen 1989).

[3] These proposals are focused on integrated CASE environments, whereas we are interested in CASE shells.

[4] By a methodology, we mean systematic principles to develop an IS such as ISAC (Lundeberg, Goldkuhl and Nilsson 1980) or IE (J. Martin 1988).

[5] This can be regarded as a set of "things'" about which we need to store information in order to model the information system in the organization adequately.

[6] The OPRR datamodel was first introduced by Welke (1988) and is defined formally by Smolander (1991b). It has been used in some CASE shells such as Meta-System's *QuickSpec* (Welke 1988), LBMS's *Information Manager* (1991), and the *MetaEdit* prototype (Smolander et al. 1991).

[7] A graph can also act as an object. In graph theory, a similar situation appears when some node represents a subgraph.

[8] The concepts can also have tabular representation definitions.

[9] We have specified GOPRR here semiformally and without update semantics. The more formal and detailed specification is under development and needs to be completed before we implement the information architecture in a tool.