Integration of Organization and Information Systems Modeling: An Object-Oriented Approach

Minder Chen and Jay F. Nunamaker, Jr.
Department of Management Information Systems
College of Business and Public Administration
University of Arizona, Tucson, AZ 85721
602-621-2748

Abstract

This paper presents an object-oriented approach to integrating organization and information system modeling. Characteristics and applications of object-oriented systems are first reviewed to show the evolution of the application of an object-oriented approach from an implementation and programming level to conceptual modeling of organization and information systems. The need to integrate organization and information systems is then discussed. MetaPlex, a metasystem implemented in Smalltalk language to support high-level object-oriented modeling is described in detail. Case studies defining Critical Success Factors in MetaPlex and creating a Structured Electronic Brainstorming System to support collaborative work are used to demonstrate the use of MetaPlex in integrating organization and information system modeling at both individual and group levels. Directions for future research are also discussed.

1 Introduction

In this paper, the authors will address the issues related to integration of organization and information system modeling as well as the development of automated tools to support object-oriented modeling. The possibility of using the same object-oriented approach in various phases of system development (including organization and information systems) promises a cohesive integration of organization and information systems. In Section 3, we will first establish arguments for the necessity of integrating organization and information system modeling. The architecture, design, and implementation of an object-oriented metasystem called MetaPlex which supports organization and information system modeling at a very high level are discussed in Section 4 and Section 5. Section 6 presents case studies defining Critical Success Factors in MetaPlex and creating a Structured Electronic Brainstorming System to support collaborative work to demonstrate the use of MetaPlex in integrating organization and information system modeling. The paper is concluded with suggestions for future research.

2 The Object-Oriented Approach and Its Applications

The term "object-oriented systems" has have different meanings for different people [Stefik and Bobrow, 1985]. However, there is general agreement that an object-oriented system should have certain characteristics. In this section, general characteristics of object-oriented systems are first reviewed and then a discussion of application areas using the object-oriented approach is presented. Concepts, entities, and things are represented as objects in an object-oriented system. Objects have structures and behaviors. The static structures of an object are described by its private data (i.e., instance variables) which store the local status of the object. The dynamic behaviors of an object are simulated by its private methods (i.e., indirect procedure calls). Objects interact with each other collaboratively to accomplish required tasks. A message has to be sent to an object in order to access or change its status. This message passing paradigm is borrowed from the human communication model [Ingalls, 1981]. Objects can interface with the external world only through a set of predefined message patterns (i.e., protocol). In most object-oriented languages, direct access to an object's private data is prohibited. The message passing paradigm facilitates information hiding, thereby increasing the modularity, modifiability, expandability, and maintainability of a software system [Klahr, etc., 1986]. Different classes can share the same message pattern to support polymorphism. For example, the message rotate can be sent to instances of Square and Triangle but these objects will have different implementations of their instance methods for responding to the message rotate. A message received will be responded to according to the message receiver's designated method. Polymorphism thus reduces the syntactical complexity of the system and allows the development of more generic codes.

Objects are organized by class hierarchy (e.g., in Smalltalk, Circle is a subclass of Arc, Arc is a subclass of Path, and Path is a subclass of DisplayObject). An object can inherit structures (instance variables) and behaviors (methods) from its class and superclasses. Subclasses can implement a method having the same message pattern to override the generic behavior defined by their superclasses. The class hierarchy promotes both data and procedure abstraction in a software system, which makes the resulting system more reusable.

2.1 Characteristics of Object-Oriented Systems

2.2 Application Areas of the Object-Oriented Approach

The object-oriented approach has been recognized as a unified paradigm in the design and implementation of programming languages, knowledge-based systems, databases, and human interface design [Zaniolo, etc., 1986]. Table 1 shows the progress of an object-oriented approach in various application areas (listed loosely in chronological order according to when the object-oriented approach was first applied):

1. Simulation. The area to which the object-oriented approach was first applied was simulation, as in the Simula...
5. Database systems. The development of object-oriented database systems is being explored because of the persistent problems in storing large amount of data in an object-oriented system as well as requirements to represent complex objects in CAD/CAM and CASE applications. Iris [Fishman, et al., 1987] and GemStone [Maiar and Stone, 1987] are examples of object-oriented database systems.

6. Knowledge base systems. Recent developments in commercial expert system shells also show the artificial intelligence (AI) industry's endorsement of the object-oriented approach to knowledge representation. For example, KEE demonstrates how object-oriented tools can be used for knowledge-based system building. It is a hybrid AI tool which integrates rule-based systems, frame-based systems, graphics, and active values (i.e., demons) with an object-oriented kernel [Kunz, Kehler, and Williams, 1984]. NExpert [Neuron Data, 1987] is a high-end hybrid rules and object-based expert system environment with an object-oriented architecture.

7. System analysis and design methods. In an object-oriented system analysis and design method, the decomposition of a system is based on the concept of objects [Botch, 1986]. Object-oriented design combines data structure-oriented, procedure-oriented (i.e., data flow-oriented), and architectural design because objects carry the notions of data structures and procedures (i.e., methods) while, at the same time, message passing among objects defines the interface of the system architecture [Pressman, 1987].

8. Organization modeling. The information processing model was developed by Galbraith [1977] as a framework for organization design. Blanning [1987] extended this information processing paradigm by applying an object-oriented approach to the modeling and simulation of an organization. Blanning [1987] also believed that an object-oriented approach could be applied to the analysis of organizations as well as to the design and implementation of systems (e.g., information systems, decision support systems) that support those organizations.

Using the object-oriented approach in information system analysis and design, as well as in organization modeling, extends the use of the object-oriented approach to conceptual modeling, which is far beyond the original notion of objects at the implementation and programming levels. This shift in focus demonstrates the generality of the object-oriented approach. However, Blanning neither addressed the issues of integration of organization and information systems nor demonstrated how tools can be developed to support the modeling process. Bosch [1986] also showed concern for the way tools can be used to support object-oriented development and called for further research in this direction.
The popularity of object-oriented systems arises from the belief that the notion of an object reduces the semantic gap between the model and the real world. In this research, the authors address issues related to the integration of organization and information system modeling as well as the development of automated tools based on object-oriented modeling. The possibility of applying the same object-oriented approach in various phases of system development (including organization and information systems) promises a cohesive integration of organization and information systems.

3 The Need to Integrate Organization and Information System Modeling

Information systems are used to facilitate the decision making and communication processes within an organization and have recently been used by companies as strategic weapons against their competitors [Ives and Learmonth, 1984]. Information systems planning methods such as Critical Success Factors (CSF) [Rockart, 1979] and Business Systems Planning (BSP) [JRM, 1984] have started to address the importance of business objectives to the determination of information system requirements. Both BSP and CSF use interviews to involve managers in the information systems planning process. However, none of these models can adequately represent the couplings between organization and information systems models. Only when the design and development of organization and information systems are tightly connected can managers quickly respond to the changing business environment and take advantage of new IS technologies.

The lack of integrated methods and software tools is one of the major causes of the missing link between organization (i.e., business systems) and information systems modeling. An integrated environment should allow users to specify the complicated linkages between business systems and information systems so that any changes in the information systems can be reflected in their corresponding business counterparts, and the dynamics of the business environment can be propagated to the supporting information systems.

The quality of any system under development is bounded by the available languages and tools as well as the way people use them [Lyytinen, 1985]. The broad scope and the rich semantics needed to describe organization and information systems exclude the possibility of using traditional database systems, structured methodologies, or simulation languages for support of object-oriented modeling. A metasystem has the flexibility to allow users to define their own terms to describe a target system [Demetrovics, Kauth, and Radu, 1982; Kottemann and Kononyski, 1984]. The generic nature of the knowledge representation scheme employed by a metasystem allows users to define customized languages for the specifications of both organization and information systems. The flexibility of a metasystem approach reduces the semantic gap between the specification tools and the application domains, thereby facilitating user learning and acceptance of specification tools that are generated.

The disadvantage of most existing metasystems results from their poor interface design. The syntactical complexities of their meta language also make it difficult to define a new language. A successful organization modeling tool must allow managers to use it to model their own business environment. MetaPlex is an object-oriented metasystem developed to demonstrate that such a system can be used in a wide spectrum of application domains and at the same time can be easily learned and used by end users.

4 The Knowledge Representation and Architecture of MetaPlex

The development of MetaPlex was inspired by PSL/PSA, a computer-aided tool for system analysis and documentation [Teichroew and Hershey, 1977]. MetaPlex is an object-oriented metasystem which can generate system specification tools for both information and organization system modeling. Systems generated by MetaPlex are intended to be used by managers and end users. Representing and manipulating relations and the design of friendly user interface have been the focus of MetaPlex development.

4.1 Knowledge Representation in MetaPlex

A system can be defined as a group of related components which interact to achieve a high level objective of the system as a whole. Both organizations and information systems are instances of generic systems. To describe an existing system or to design a new system, system analysts or designers are primarily concerned with system components and relationships among them. Various knowledge representation schemes have been studied to see whether they can be used for system specifications. A rule-based system can be used for capturing the designer's know-how on certain design decisions. A frame-based system can be used to represent general design schemata [Lubars and Harandi, 1986]. However, neither of these can explicitly represent the interrelationships of objects in a system. The knowledge representation scheme used in MetaPlex is based on a three-level abstraction of an object-oriented model: the axiomatic, median, and instance levels [Kottemann and Kononyski, 1984].

As shown in Figure 1, Object Types, Relation Types, and Attribute Types are built-in knowledge representation primitives of MetaPlex at the axiomatic level. At the median level, a domain specific language can be defined by using Object Types and Relation Types with a set of Attribute Types identified in a domain. An Object Type has its name, a comment, and a set of Attribute Types to characterize it. Relation Types specify the relations among Object Types. An Attribute Type is characterized by a name, a data type, legal values, a default value, and occurrence (e.g., #one or #many). The data type can have the following data types: Integer, Real, Boolean, String, and Description. Currently the Description data type can be used to encode unstructured and procedural knowledge in text format. The structural knowledge of the domain in general is thus captured at the median level as a system description language. Prototypical knowledge of a domain captured at the median level will be used to guide users in defining a target system specification at the instance level. At the instance level, a target system is defined by a system description language in terms of objects and relations with attributes.

4.2 Relations and Cross Referencing in MetaPlex

In MetaPlex relation types are defined as a Smalltalk class so that relation types can have their own attribute types, can be attached with procedures, and can form inheritance hierarchy. Currently, only attribute types can be used to describe a relation. In MetaPlex a relation type is defined as two, or more than two, groups of object types connected by connectors. The internal, graphical, and external language representation of a relation type are depicted in Figure 2.
4.3 Representing Abstraction in MetaPlex

Researchers have identified three major abstraction mechanisms for describing a target system: Classification, Generalization/Specialization, and Aggregation [Gibbs, 1985]. The equivalent representations of these abstraction mechanisms in MetaPlex, discussed below, demonstrate the expressive power of MetaPlex knowledge representation scheme.

1. Classification. The “object type” and “object” in MetaPlex are equivalent to “class” and “instance.” The properties and relationships defined for an object type are used to elicit information about objects of this type in a target system. For most applications, two levels of classification are sufficient [Mylopoulos, et al., 1980].

2. Generalization/Specialization. The class hierarchy in an object-oriented system can be represented by an “AKO” relation among object types. For example, we can define “REPORT” is-a-kind-of “DATA”, and “MONTHLY REPORT” is-a-kind-of “REPORT”. Property inheritance along the class hierarchy can be handled by an inference engine.

3. Aggregation. There are two types of aggregation: Cartesian aggregation and cover aggregation. In MetaPlex, Cartesian aggregation means that an object is an aggregation of its attributes. The cover aggregation can be specified by using a decomposition relation in the following format:

\[
\text{GROUP-ITEM} \text{ consists-of } \text{[GROUP-ITEM, DATA-ELEMENT]} \\
\text{GROUP-ITEM, DATA-ELEMENT} \text{ is-part-of } \text{[GROUP-ITEM]}
\]

Language definers can define a relation type as one-to-one (list structure), one-to-many (tree structure), or many-to-many (network structure). Complicated relations can be easily represented in MetaPlex.

4.4 The Architecture of MetaPlex

The design goal for MetaPlex is to develop a simple, but flexible, computer-aided system specification tool that can support and be applied to various domains. The ease of use of MetaPlex is achieved through an interactive menu-driven user interface and a graphic representation of a target system description.

While other metasystems use the compilation approach [Yamamoto, 1981; Demetrovics, et al., 1982], MetaPlex uses the interpretation approach for language definition and target system specification. The interpretation approach makes it much easier to develop description languages and to experiment with them. Eventually, users will be able to develop languages of their own without any help from language definers.
The architecture of the MetaPlex System is shown in Figure 3. It has three subsystems: Language Definition System, System Description System, and Transformation System. At the meta level, language definers can use the Language Definition System to define system description languages and their consistency and completeness checking rules. Language definers can also use the Transform System to define languages for the transformation of system descriptions from one language to another. Language Syntax Report and Consistency and Completeness Checking Rules Report can be generated by the Language Definition System. These reports also can be used by language definers to check the completeness of the language defined or given to description definers as a user manual of a system description language.

At the instance level, description definers can use the System Description System to define a target system description. Report facilities, on-line query functions, and a Structure Browser can be used by description definers during the specification process to verify an existing system description. Consistency and completeness checking rules defined for a system description language can be applied to check the consistency and completeness of its system descriptions. The Consistency Checking Report and the Completeness Checking Report can be generated and used by the description definers. Description definers can also use an appropriate transformation language defined by language definers to convert a system description in one language into a system description in another language. The detailed design of MetaPlex is discussed in Section 5.

In MetaPlex, some functions at the meta level are made available to description definers. For example, description definers can formulate a simple completeness checking rule and check incomplete objects on-line. They can also create a dynamic sublanguage from an existing language and then open an editor to interact with a subset of a system description. Making some meta level functions available to description definers provides them with additional flexibility to fulfill unique requirements of a target system.

5 The Design and Implementation of MetaPlex

In this section, the design and functionalities of Language Definition System, System Description System, and Language Transformation System are discussed. Implementation issues are also addressed.
IF the object type of an object = DATA STORE
   the object is not involved in relations
   [DATA STORE] generates: [DATA FLOW] received-by: [PROCESS] AND
   [DATA STORE] received: [DATA FLOW] generated-by: [PROCESS]
THEN incompleteness type = 'DATA STORE is not generating
   and receiving any DATA FLOW'

Table 2: A Completeness Checking Rule

Figure 6: The Structure Browser for Relations in a System Description

The upper pane in the Attribute Browser window presents a
scrolling set of attributes of an object or a relation selected from
the Target System Editor. Users can switch the contents of
the lower pane in the Attribute Browser to display either the
definition or the value of the selected attribute. A question mark
(?) is concatenated to the end of an attribute name to inform
users that the value of the attribute has not yet been defined.

The Formatted System Description Report can be generated
from the system with all the relations cross-referenced. Com-
pleteness checking rules about a system description can be de-
defined at the language level. An example of a completeness check-
ing rule is shown in Table 2. This rule stated that any DATA
STORE in a Data Flow Diagram description which has not gen-
erated or received a DATA FLOW is an incomplete object. Users
can generate a Completeness Checking Report in a batch mode
with all the incomplete objects listed. An on-line completeness
checking function is also available. Users can formulate a sim-
ple completeness checking rule on-line and let the system find all
incomplete objects according to the dynamic formulated rule.

A Structure Browser, as shown in Figure 6, is designed to
enable users to examine the relations associated with an object
in a system description. Both tree and network structures of a
relation can be examined by using the Structure Browser. A user
can select an object from the Structure Browser to examine one
type of its relations with other objects. After further develop-
ment, users will be allowed to create objects and relations in the
Structure Browser dynamically.

5.3 Transformation System

Top managers are concerned about organizational goals and objec-
tives, their organization's competitors, and strategic assump-
tions. Information systems developers are concerned with the
to detailed information required for the design and implementa-
tion of the target systems. In developing a system, its potential
users are allowed to use different languages to describe a system from
multiple aspects during different phases of the development cycle.
Part of the information captured in one phase should be trans-
ferable to another phase by using a different language. There are
two types of information in a target system description: objects
and relations. Since objects and rules are different in their na-
ture, we have decided to use two types of rules to represent the
translation of objects and relations: object translation rules and
relation translation rules.

Figure 7 shows the design of the Transformation System.
Language definers can use the Transformation Language Defini-
tion System to define a transformation language between any two
existing system description languages (called source language and
target language) as long as there is a logical mapping between de-
scriptions in these two languages. The transformation language
defined can then be used by the System Description Transfor-
mination Language, which can convert objects and relations described
in the source language into objects and relations described in
the target language. Figure 8 illustrates a Transformation Language
Editor in which object translation rules (shown in upper half
panes) and relation translation rules (shown in lower half panes)
have been defined to translate a BSP system description into a
PSL description. If the source of a converted object or a rela-
tion is recorded a system description can be traced across various
phases of its life cycle by MetaPlex. Traceability is one of the
major factors contributing to the successful integrating of orga-
nization and information systems because it allows users to trace
back from the information systems to the organization systems
that they serve.
5.4 Implementation Issues

Object-oriented languages, such as Smalltalk, promote fearless programming [Diederich and Milton, 1987]. Because of its rapid prototyping and highly interactive and graphical interface, Smalltalk was our choice for implementation. An IBM PC/AT version of the Smalltalk-80 language has been used to develop the MetaPlex. Smalltalk allows dynamic binding so that the changes in data structure design are much easier [Goldberg, 1984]. The Smalltalk user interface framework has been used in MetaPlex to provide a consistent user interface at both language definition level and system description level.

6 Case Studies Using MetaPlex

In this section two cases of using MetaPlex are presented. The first describes use of a tool generated by MetaPlex to support an object-oriented approach to extending Critical Success Factors, an information systems planning method, and thereby integrating organization and information system modeling. In the second case, the authors propose the use of MetaPlex to create a Structured Electronic Brainstorming System (SEBS) for use as a metasystem driven tool for computer-supported collaborative work (CSCW). SEBS can be used as a front-end tool to facilitate the group dynamics in a face-to-face meeting where people from various areas are involved in a system modeling task.

6.1 Defining a System Description Language for Information Systems Planning: Critical Success Factors

Critical Success Factors is an information systems planning method used to identify the key factors needed by a manager to be successful in the business. The original CSF provided only procedures for conducting the CSF study and focused only on information requirements for an individual manager's decision [Henderson et al., 1984]. Identifying CSF based only on an individual manager's needs made it necessary to make frequent changes in the resulting information systems. The authors have extended the CSF methodology to allow managers and IS personnel to develop an information system that incorporates the organization's and the manager's perspectives. Object types in the extended CSF methodology are: CRITICAL SUCCESS FACTOR, MEASURE (to the CSF), REPORT or QUERY (to reflect these measurements), INFORMATION SYSTEM (which generates the reports and queries), BUSINESS GOAL (behind these critical factors), and ORGANIZATION ENTITY (involved). All these object types and the relation types among them have been defined in MetaPlex as a language so that the System Description System in MetaPlex can use that language to guide users when they apply the CSF methodology.

The CSF methodology for information systems planning follows a four-step procedure [Martin, 1983]:
1. Identify the critical success factors. Managers as a group are asked to identify the critical factors which will affect the success of their business.
2. Determine the measurements for identified critical success factors. When critical success factors are difficult to quantify, soft measures must be used. 3. Design reports or on-line queries to inform managers of the status of the CSF being measured or changes that have been made. Reports or queries must be generated from an existing or a new information system. The managers will use these reports or queries as not necessary the same managers who identify CSF or define the measures for CSF. The original CSF methodology is designed to be used by individual executive officers to define their own information needs, but the method can be extended for use by a group of managers to determine the information system requirements of an organization as a whole. A GDSS can be used to support groups who use CSF to define their information needs.

Figure 9 is a representation model for the Critical Success Factors methodology. A representation model is a graphical representation of the object types and relation types defined by a method. In such a representation model, object types are enclosed in rectangles, relation types among object types are represented as labeled arrow lines (only binary relationships can be represented in a representation model and only one direction of a relation is shown in Figure 9). To simplify the presentation of the model, attribute types that describe object types and relation types are not shown. As illustrated in this figure, the extended CSF method contains object types in both information systems and organization systems domains. It can be used to bridge the gap between information systems planning and business planning. Language definers can extend the language to cover other related object types and relation types in both organization and information systems domains. The MetaPlex System Description System can use the language defined for CSF to generate a tool for description definers to use.
6.2 An Example of Generating a GDSS Tool from MetaPlex: Structured Electronic Brainstorming System

Christakis [1987] noted that "inter-disciplinary teams cannot work productively and efficiently in designing complex systems unless their work is supported and argued by methodologies that have been invented specifically for this task." This concern is especially relevant for large software projects in which members with diverse backgrounds need to work together. A metasystem approach can be used to provide the flexibility to customize methods for both group deliberation in GDSS and system definition in CASE.

Groups who use Plexsys in meetings generally use the following process: A meeting agenda is set before the meeting. Meeting participants first use the Electronic Brainstorming (EBS) to generate ideas related to the question posted on a screen at their PC workstations. Ideas created (also called EBS comments) are randomly sent out through a local area network to other participants to stimulate them to generate more ideas. An EBS session usually takes 45 minutes. Following the EBS session, an Issue Analyzer (IA) tool is used to identify and consolidate the EBS comments into major issues. A callalive voting program can be used to prioritize the consolidated issues. The use of Issue Analyzer usually takes about 60 minutes. Since ideas are entered as free format text, an idea generated by one participant can be interpreted by another outside its original context. Structured Electronic Brainstorming System (SEBS) has been designed to eliminate the possible loss of structures and relations embedded in the original ideas during the group process.

Software development meetings usually are held for specific purposes, such as to define the system objectives, to identify all the required reports and queries, or to determine system functions. These meetings employ some structured methods, such as Joint Application Design [IBM, 1987]. Before running an SEBS session, the session leader will work with some key participants to choose or define a MetaPlex system description language for the meeting discussion. A language for SEBS is defined based on the nature of tasks, the common terminology used in an organization, theories and methods in a specific domain. Once a language for SEBS has been defined, it can be relied upon in place of the facilitation skills required of a session leader for a similar meeting.

The flexibility of the metasystem has expanded Plexsys Planning Tools into a GDSS generator which can generate customized GDSS environments to support a wide-range of tasks. This extension to GDSS capability, as Huber suggested, will increase the possibility of successful adoption of a GDSS in an organization [Huber, 1984]. The session leader should define "the language of business" to reduce the communication barrier among meeting participants [Martin, 1987]. Since users and managers are allowed to "speak their own language," they will be able to participate fully in the discussion.

The language specified by the session leader will be loaded into the SEBS tool to systematically direct the contents and structures of group discussion. Whenever users want to generate a new idea, the SEBS tool asks them to categorize it according to the object types defined in the language. Users can also create relations among ideas, such as specifying "Group Item A can be decomposed into Data Item B, C, and D," "Report E should include Group Item A and F," etc. Detailed attributes about objects and relations, such as "the data format of Data Item is 9(X)," can be collected through the Attribute Browser.

Additional advantages of using the SEBS tool are its capability to:

1. Merge the brainstorming and issue identification processes. By merging the original brainstorming and issues analyzing processes SEBS helps to reduce meeting time.
2. Allow users to generate ideas on several related issues at one brainstorming session. Instead of brainstorming on a single question such as, "What are the critical success factors of application ABC?", the system allows users to generate ideas on several categories. An EBS question can be as complex as "What are the Input, Report, and Process in application XYZ?" and "What are the relationships among Input, Process, and Report in application XYZ?"
3. Help users focus on the issues under discussion. Current Plexsys tools are adequate for exploration type meetings. However, to collect specific information for system development in a meeting, SEBS will be an ideal tool.
4. Facilitate the acquisition of knowledge concerning complex relations. In the current Plexsys Planning Tools, Issue Analysis supports the identification and consolidation of issues from an EBS file. Relations among issues usually get lost in the consolidation process. SEBS allows participants to enter relations among objects in a structured format to prevent this process loss.
5. Integrate SEBS results into other modeling tools. Information captured has been categorized in a structured format so that SEBS meeting results can be easily transformed and exported to other modeling tools, such as CASE tool [Chen and Nunamaker, 1988].
6. Apply completeness and consistency checking on meeting information. Just as Problem Statement Analyzer can be used to analyze information in a PSA database captured through Problem Statement Language, similar analysis functions can be applied to meeting information, since it is captured in structured form. The meeting participants will be informed of missing information and of inconsistencies in their argumentation.

The SEBS tool is just one example of using a metasystem to drive a CSCW tool. Currently Topic Commenter in Plexsys [1988] has very limited metasystem features. A facilitator can create a list of topics so that participants can make comments on each topic. Each topic considered in a session is like an object type. The current Topic Commenter does not have the capability to elicit attributes for instances in each topic and does not support the description of relationships. SEBS provides its users with structure and flexibility so that it can be used as a front-end tool to facilitate face-to-face meetings for system modeling involving many participants.

7 Conclusion

The future success of information systems in an organization will depend on how information systems development can be tied in with business systems. In this paper we have presented an object-oriented metasystem approach which allows us to generate a software environment for integrating organization and information system modeling. The relative usability of MetaPlex and other manual and automated tools calls for some empirical stud-
ies, through which future research directions will be suggested. The use of MetaPlex to generate GDSS tools as front-ends to collaborative modeling opens up many opportunities for further investigation.

8 References


