New Jersey Institute of Technology

Digital Commons @ NJIT

10-31-1992

Domain analysis within the GenSIF framework

Heiko Thimm
New Jersey Institute of Technology

ABSTRACT

Domain Analysis Within the GenSIF Framework

by
Heiko Thimm

The GenSIF framework which is targeted towards very large, distributed, and complex software systems recently has been proposed to accomplish a form of systems engineering and systems development in which the issue of systems integration is considered from the beginning on.

One of the components of GenSIF is domain analysis. Domain analysis leads to the design of a domain model. The specific needs GenSIF has in that area were investigated with an emphasis on domain modeling. Main points addressed in that investigation were the issue regarding the relevant information for the domain modeling process and the required type of domain model.

Based on these results, an approach to domain modeling for GenSIF was developed that provides a specific graphical notation which allows to create a semi-formal kind of domain model. A few modeling examples for the application domain "university department" were designed to evaluate this notation.

In addition, the major aspects of the application of a computer based tool with respect to domain analysis as a concept of GenSIF were analysed.

DOMAIN ANALYSIS WITHIN THE GenSIF FRAMEWORK

by
Heiko Thimm

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science
Department of Computer and Information Science
October, 1992

APPROVAL PAGE

Domain Analysis Within the GenSIF Framework

by
Heiko Thimm

_6/22/1992_

—————————————————————
Dr. Wilhelm Rossak, Thesis Adviser
Assistant Professor of Computer and Information Science, NJIT

_6/22/92_

—————————————————————
Dr. Peter Ng, Committee Member
Chairperson and Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Heiko Thimm

Degree: Master of Science in Computer and Information Science

Date: October, 1992

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science, New Jersey
  Institute of Technology, Newark, NJ, USA, 1992

- Diplom-Informatiker (FH), Fachhochschule Konstanz
  (Polytechnic University), Konstanz, Germany, 1991

Major: Computer and Information Science

Missing Page

ACKNOWLEDGMENT

The author wishes to express his gratitude to Professor Wilhelm Rossak, for his faithful supervision, friendship, and constant moral support throughout this research.

Special thanks to Professor Peter Ng for serving as a member of the committee and for the time invested on this research.

The author especially is grateful to Tamar Zemel for helpful discussions, valuable suggestions, and constant support throughout this research.

The author is grateful to the Fulbright Commission for the opportunity to complete the Master of Science Program in the United States and for funding provided through their scholarship. Mostly the author appreciates the knowledge gained through experiences in both academic and personal aspects of student life in the United States.

TABLE OF CONTENTS

Missing Page

Missing Page

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

A new generation of computer based systems with fundamental differences to the traditional ones is recognizable. The specific characteristics of these new systems prevent the successful application of engineering methodologies that have been successfully applied to the development and management of traditional systems. Especially for their development as integrated systems, these new systems necessitate the innovation of new engineering methodologies and frameworks.

Currently such a framework is under development at the Institute for Integrated Systems Research at the New Jersey Institute of Technology. This framework has been called, due to its generic character, "Generic Systems Integration Framework" or in its abbreviated form just GenSIF.

Integration architectures are an essential component of GenSIF. One major aspect is that such an integration architecture must fit to the given environment, the application domain, since integration architectures are domain specific.

For the decision about the fitting integration architecture, domain analysis is a prerequisite that has been included as a component of GenSIF. Domain analysis which goes beyond just requirements analysis is a model driven analysis approach with the goal to derive and maintain a model of the application domain that is called a "domain model". Besides its utilization in the decision process concerning the fitting integration architecture, this domain model is also the input for the requirements analysis of each application project within the domain.

In other words the domain model that is designed and maintained in domain analysis, has a specific role in systems engineering as well as systems development with GenSIF.

This thesis has been prepared as a contribution to the GenSIF project, with respect to its domain analysis component. The core of the thesis is organized in two main parts, where part one consists of the chapters 2 to 5 and part two includes

the chapter 6 and 7.

Part one is concerned with the elaboration of some major aspects of domain analysis as a concept of GenSIF. After GenSIF has been introduced in general, the different roles of the domain model within GenSIF are investigated. Then an overview about the general domain analysis research area is provided. In the last chapter of the first part, some important aspects of the process of domain analysis are discussed, considering and discussing the specific needs GenSIF has in this area.

Based on the results of the first part, in the second part an approach to a specific graphical notation for domain analysis within GenSIF is proposed. In the remaining chapter 7 of the second part some remarks regarding a domain analysis support tool for GenSIF are summarized.

At the end of this thesis a final discussion is provided. In this discussion first a summary of the research this thesis is concerned with is provided before open research questions with respect to each of the addressed areas are discussed. This discussion also includes shortcomings of the introduced specific modeling notation. Finally conclusions and recommendations regarding the continuation of the research efforts of this thesis are provided.

CHAPTER 2
AN OVERVIEW OF GenSIF

In the following chapter an overview of the GenSIF framework is given. Although the documentation prepared by the inventor of the GenSIF framework has been studied carefully, it should be pointed out that the overview is based on the authors point of view. Hence the reader is referred to the original documentation for more information (Rossak and Ng 1991; Rossak 1992a; Zemel 1992).

2.1 What is GenSIF?

If we think about the usage of computers in organizations of our society like business organizations, universities, federal organizations, production plants, or even non profit organizations, in general we can conclude that almost every task is done by the usage of computer systems. Most of these traditional systems can be characterized in the following way (Zemel 1992):

- specific user group

- one purpose

- small size

- short life

- homogeneous environment

- each part depends on other parts.

Based on a new approach called global thinking, buzz words like Computer Integrated Manufacturing (CIM), Computer Integrated Business (CIB) or even Computer Integrated Industry (CII) emerged over the last few years. We also have

some research groups that work in the field of Computer Based Systems Engineering (CBSE). These new approaches all indicate that we are approaching a new generation of computer systems. Some of the main characteristics of these new systems are:

- no specific user group

- more than one purpose

- large and complex

- long life

- heterogeneous environment

- each part is a system by its own

A major issue of that new generation of computer systems is integration on the systems level. But integration in that sense does not mean melting components in one big system. A new system type, called mega-systems, is what is required for that new generation of computer systems (Zemel 1992).

In a mega-system, the components are loosely coupled components, where each element is still self-contained, but all elements interoperate. A mega-system should be as open as possible like a general system but as preplanned as possible as well (Rossak 1992b).

One of the major problems of traditional systems development is that maintenance becomes close to impossible. In the worst case the existing system has to be phased out and redeveloped from scratch. In contrast to that, if a component of a mega-system is replaced or upgraded, this should have no direct effect on the other components. Based on this problem and other aspects of development/maintenance, the conclusion is that traditional systems development approaches are not adequate for that new generation of systems (Zemel 1992).

The so-called Generic Systems Integration Framework, abbreviated GenSIF (Rossak and Ng 1991), which is an on-going research project at the Institute for Integrated Systems Research at the New Jersey Institute of Technology under the leadership of Professor Dr. W. Rossak has been proposed as a possible solution. GenSIF, as the name suggests, is a generic framework for systems integration for very large, complex, distributed systems that contributes to systems engineering and systems development of that new generation of computer systems. The focal point of GenSIF is the engineering aspect where in (Zemel 1992) a process model that is based on GenSIF is introduced.

The underlying believe of GenSIF is, that mega-systems are not developed in only one project, but by many projects. All activities of the development process within the application domain are integrated by modeling the domain in a domain model and deriving an integration architecture.

The integration architecture is divided in a conceptual architecture model, providing guidelines and standards, and in an infrastructure, providing the environment for development and usage of such an integrated system. If everything is build according to these guidelines and the infrastructure, the resulting system can be used as one integrated system (mega-system).

Using these components of the integration framework, a meta-level of control for system development can be specified. This meta-level provides the system engineer with a basis to coordinate and to plan projects in the application domain (Rossak 1992a).

2.2 The Components of GenSIF

Figure 2.1, adopted from (Rossak 1992a), gives an overview about the components of GenSIF. These three components reflect different levels of abstraction and address different goals and needs during an integrated development process.

Each of these components is introduced in the following paragraphs, based on the

Global (Domain) Integration

INTEGRATION

ARCHITECTURE

ENABLING TECHNOLOGIES
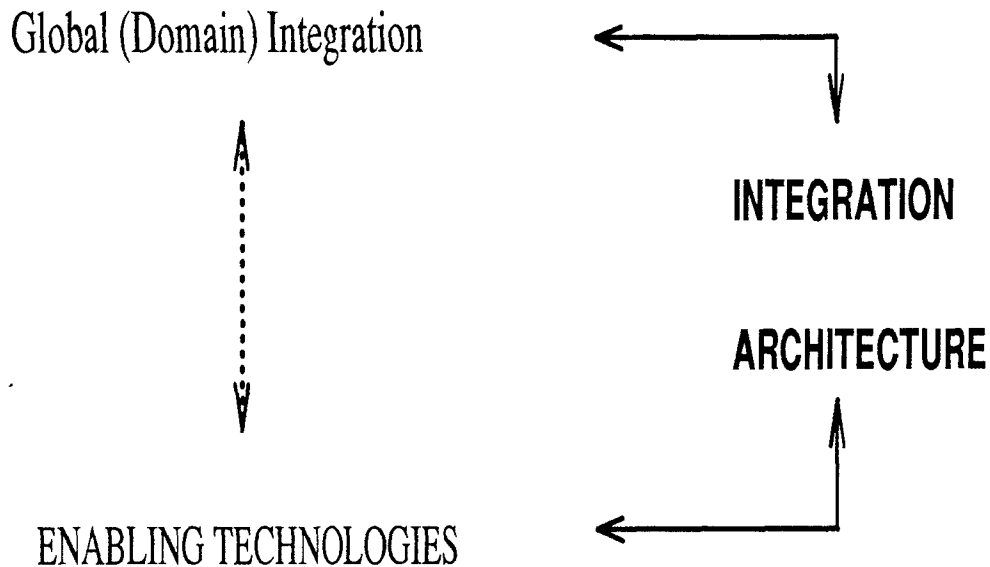
Figure 2.1: The components of the GenSIF framework.

definitions given in the papers which are intended to make the GenSIF framework
public to the system engineering and software engineering community (Rossak and
Ng 1991; Rossak 1992a).

Global (Domain) Integration:

Global (domain) integration specifies the conceptual basis for the integration archi-
tecture. One aspect of global integration is to deal with the concepts and semantics
of an application domain and with the mapping of these concepts into the installed
applications. These global activities involve an analysis of the application domain
in order to define a common model of the environment the system is going to serve.
Domain analysis (Prieto-Diaz, and Arango 1991) not only provides a basis for se-
mantic integration, but it is also the main input to decide on the design of the
integration architecture.

Integration Architecture:

The integration architecture is the core of GenSIF. An integration architecture is
a conceptual model that bridges the gap between the results of domain analysis

and the tool-level. It also is an infrastructure which provides the necessary utilities and components to implement an application system by following the rules of the conceptual model. An integration architecture must fit the needs oft the application domain, like a given hardware- architecture must fit the needs of the typical working environment it is serving.

Enabling Technologies:

Enabling technologies comprise all the tools and products that are required by the infrastructure of an integration architecture to develop and implement the applications which will fill the abstract architecture with functionality and data. This level should not only be concerned with the state-of-the-art but should also provide suggestions for restrictions and standards in this area.

The above identified components of GenSIF provide the necessary models to handle strategic decisions and technical integration issues. The components of GenSIF affect and guide the development of all system parts. These system parts are usually developed in separate, independent projects.

CHAPTER 3
THE ROLE OF THE DOMAIN MODEL
IN GenSIF

After having introduced the general idea of the GenSIF framework and its compo-
nents, this chapter concentrates on the utilization of the domain model in GenSIF.
It is supposed to provide the background information that is presupposed by the
succeeding chapters and completes the description of the GenSIF framework on an
introductory level. The reader is referred again to the original documentation of
the GenSIF framework for more information (Rossak and Ng 1991; Rossak 1992a:
Zemel 1992).

3.1 The Role of the Domain Model in
Systems Engineering
With GenSIF

According to GenSIF, systems engineering consists of two phases (Rossak, and
Prasad 1991). During the first phase, the integration architecture is developed.
During the second phase, the architecture is used for global integration and for
evaluation of enabling technologies. With respect to the investigation of the role
of the domain model within systems engineering according to GenSIF, we can
concentrate on the first phase. As shown in figure 3.1, before a new integration
architecture can be designed or an existing one is adopted, domain analysis must
be completed (Rossak 1992a). Domain analysis as it should be performed there
is investigated in chapter 5. The focus is on the results of the domain analysis,
i.e. the acquired knowledge about the application domain which is formalized and
represented in a domain model.

Based on the domain model an integration architecture is derived. Implicitly
this has pointed out that integration architectures are always developed for specific
domains. The activity, where a fitting integration architecture is derived from

DOMAIN ANALYSIS

INTEGRATION ARCHITECTURE

GLOBAL INTEGRATION ~~- - - - - - - -~~> ENABLING TECHNOLOGIES
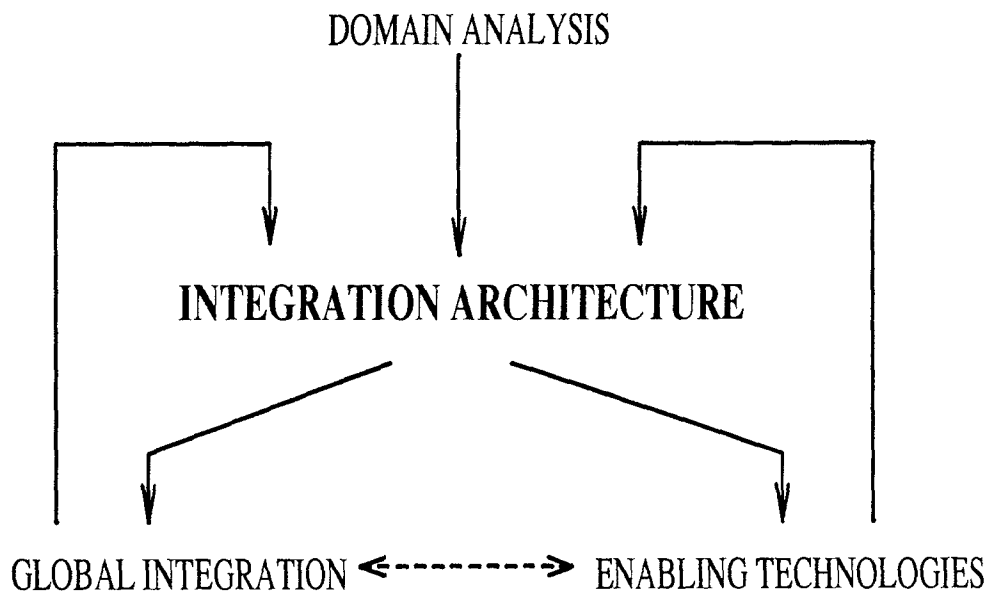
Figure 3.1: Developing an integration architecture

the domain model, has been called "mapping" the application domain model into a fitting integration architecture. It is extremely important that the "mapping-procedure" will find a fitting integration architecture, otherwise the utilization of the GenSIF framework will fail.

The integration architecture provides a conceptual and technical framework for systems development. This framework is used as a standardization and control tool for every development and every existing component in the application domain. It is used to provide a basis of reference for the projects in the application domain and gives the developers and project managers a chance to evaluate decisions, trade-offs and implications (Rossak 1992a). However the integration architecture does not speak about particular application programs.

3.2 The Role of the Domain Model in Systems Development With GenSIF

Integrated development, as it has been described in (Rossak 1992a; Rossak 1992b), typically has to cope with diversified and complex development environments where the (sub-)systems are developed in independent projects as a part of the larger development effort within the application domain. To achieve integrated development, a meta-level above the level of the single projects has been introduced in (Rossak 1992a). This meta-level is oriented towards long-term goals and control of the shorter projects, in order to assure integration of these projects and their results into the global system framework.

The meta-model proposed in (Rossak 1992a) is divided into two development tracks which reflect this meta-level and the project-level. The components of the GenSIF framework namely a domain model, an integration architecture, and a specification of enabling technologies are proposed as the instruments on the more long-term oriented meta-level. The project-level is given by the development-cycle for the subsystems of the integrated system. This approach is graphically depicted in figure 3.2 and shows the role of the domain model in system development with GenSIF (Rossak 1992a). On the project level, the domain model specifies the conceptual environment for each of the projects. It includes those concepts and terms of the "real world" which may be reflected in the application. It is used to express the project's goals and requirements. An important point to notice is that the usage of the domain model as a common reference model must be enforced for the requirements analysis of each project. Therefore, the requirements analysis is based on the domain model. The task of the requirements engineer can be described as a process where the domain knowledge, that is captured in the domain model, is refined up to a certain level. This allows to see the specific requirements of a project as a part of (and derived from) the basic structure of the domain. The

(CONTROL &
EXPERIENCE)

PROJECT(S)
LEVEL

META-LEVEL
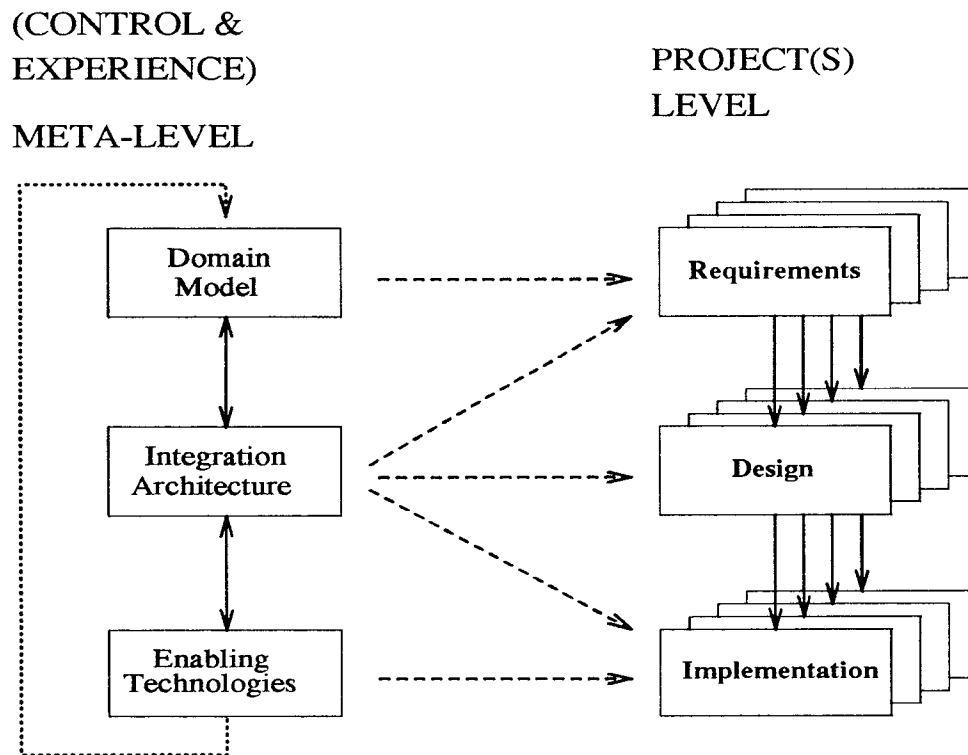
Figure 3.2: Systems development with GenSIF.

proposal for this usage of domain models is also based on the experience that large software systems are hard to change. Thus, to be able to base design decisions not only on an initial set of user requirements, but on a comprehensive model of the system's application domain, the domain model contains the objects, relationships, and concepts that are considered important to those who will use the system. This model is likely to be more stable than a set of specialized requirements.

Greenspan et al. give a good overview of the domain-modeling approach to traditional software development in (Greenspan, Mylopoulos, and Borgida 1982):

".. in considering the development of a variety of information systems we have found it necessary to become intimately familiar with a wide range of subject matters: medical knowledge, hospital procedures and policies, available therapies (drugs, surgery, etc.), legal responsibilities to government, and so on. We believe that this

kind of real world knowledge needs to be captured in a formal requirements speci-fication. The ability to efficiently design appropriate computer systems and enable them to evolve over their lifetime depends on the extent to which this knowledge can be captured".

The design method JSD (Jackson 1983) is based on the same principle:

" Every computer system is concerned with a real world, a reality, outside itself. A telephone switching system is concerned with telephone subscribers, telephone handsets, dialing, conversations, conference calls. A payroll system is concerned with employees, the work they do, the pay they earn, the tax they must pay, the holidays they are entitled to.

It is a fundamental principle of JSD that the developer must begin by modeling this reality, and only then go on to consider in full detail the functions which the system is to perform. The system itself is regarded as a kind of simulation of the real world; as replicating within itself what is happening in the real world outside. The functions of the system are built upon this simulation; in JSD they are explicitly added in a later development step".

CHAPTER 4
THE GENERAL DOMAIN ANALYSIS APPROACH

Within the previous chapters, the term domain analysis has been used several times without a further explanation of what it refers to. This was done on purpose in order to avoid the introduction of additional new terms and concepts at the same time the GenSIF framework has been introduced and under the assumption that the reader is able to intuitively understand the used term.

Since domain analysis is a research topic of its own, this chapter is supposed to give a general overview of that research area without any conclusions to domain analysis as it should be performed within the GenSIF framework. The purpose of this general overview is to prepare the elaboration of domain analysis as it is requested by GenSIF, and to identify important aspects that need to be considered in that elaboration.

4.1 The Major Concern of Domain Analysis

Let us start the description of the major concern of domain analysis with a definition of what is called a domain:

"In a broad context it is "a sphere of activity or interest: field" (Webster) in the context of software engineering it is most often understood as an application area, a field for which software systems are developed. Examples include airline reservation systems, payroll systems, communication and control systems, spread sheets, numerical control.

Domains can be broad like banking or narrow like arithmetic operations. Broad domains consist of clusters of interrelated narrower domains usually structured in a directed graph. To reserve a seat in the domain of airline reservation systems, for example, an update operation is called from the domain of database systems. To update a record in the database domain, operations from a still more basic domain,

like programming languages, are needed. Other domains like user interfaces (e.g. screen manipulation, mouse interaction) are also instrumental for airline reservation systems. Domains, therefore, can be seen as networks in some semihierarchical structure where primitive, narrow domains such as assembly language and arithmetic operations are at the bottom and broader, more complex domains are at the top. Domain complexity can be characterized by the number of interrelated domains they require to be operational (Prieto-Diaz 1990)".

Domain analysis has been defined by Jim Neighbors (Neighbors 1981) as an attempt to identify the objects, operations, and relationships domain experts perceive to be important for the domain. Diaz has extended this definition to a form, most researchers within that area seem to have agreed on today:

"... domain analysis can be seen as a process where information used in developing software systems is identified, captured, structured, and organized for further reuse (Prieto-Diaz 1990)".

It is quite plausible to draw an analogy between domain analysis and conventional systems analysis like it has been done by Neighbors (Neighbors 1981) in the early days of domain analysis research. However the important difference is that domain analysis goes beyond systems analysis. It is a meta-level version of conventional requirements analysis because of its focus on the meta-level of the software construction process (Prieto-Diaz, and Arango 1991). Neighbors has explained the difference between both approaches by identifying that systems analysis is concerned with actions in a specific system in an application area while domain analysis is concerned with actions and objects in all systems in an application area (Neighbors 1981). He therefore concluded that domain analysis can be explained as a generalization of systems analysis in which the objective is to identify the operations and objects needed to specify information processing in a particular domain. In (Prieto-

Diaz, and Arango 1991) another alternative definition of domain analysis has been introduced in which it is regarded as a form of knowledge engineering designed to support a particular problem-solving process: case-based software specification and construction.

Domain analysis is at the intersection of a family of disciplines: software specification, automatic software development, conceptual modeling, knowledge acquisition, knowledge representation, and software engineering economics. From all these areas it can draw viewpoints and solutions. From them, it also inherits difficult questions and open problems. Related domain analysis experiences such as in automatic programming, expert systems development, object-oriented software development, development of software factories and also in library science are reported in (Prieto-Diaz 1990). Despite this large spectrum of different fields where domain analysis is evident, in most published research papers concerned with domain analysis there is a strong tendency to explain domain analysis as an activity oriented towards software reusability. This tendency might come from the fact that the domain analysis process automatically leads to a domain model that allows to reuse the identified and structured information concerning the domain.

An attempt to identify concrete tasks of the domain analysis process is given in (Prieto-Diaz, and Arango 1991), where the following steps have been identified:

- **Domain Identification:**
  Identification of the boundary of the domain that defines its scope i.e. the objects, operations, and relations that belong to the domain.

- **Information Acquisition:**
  Identification, selection, and acquisition of information concerning the identified domain.

- **Domain Model Representation:**
  Making all that acquired information explicit, and readily available in a formal domain model. In reality this step goes hand in hand with the former step.

- **Evolution:**
  Typically, knowledge of a domain evolves naturally over time. This has to be reflected in the domain model.

- **Evaluation, i.e. Verification and Validation:**
  Verification demonstrates that the model is syntactically and semantically correct with respect to the definition of the modeling formalism. Validation demonstrates that the information captured by the model does indeed serve the goals for which domain analysis is applied.

Domain analysis is a broad and complex subject area. To cover all ramifications of this research area goes beyond the scope of this paper. Due to the nature of the activities and issues involved and to the newness of the area, domain analysis is perceived differently by different communities. What is shared by all different approaches to domain analysis that have been reviewed for that thesis is that domain analysis implies domain modeling i.e. the design of a model of the domain. This issue is explored in more details in the following section of this chapter.

4.2 Domain Modeling as Part of the Domain Analysis Process

The output of the domain analysis process is a domain model. According to (Iscoe, Williams, and Arango 1991) domain models within the context of domain analysis are representations of an application domain that can be used for a variety of operational goals in support of specific software engineering tasks or processes. These operational goals are always implicit in the construction of a domain model and are essential to understand the form and contents of that model. In other words, the operational goals specify the usage of the application domain knowledge in support of various software engineering tasks and processes. Typical examples for operational goals of domain modeling for software engineering are (Iscoe, Williams, and Arango 1991):

- **Requirements and Specifications:**
  Eliciting, verifying, and formalizing software requirements and specifications.

- **Automated Program Generation:**
  Generating code from a system specification.

- **Reverse Engineering:**
  Identifying the semantics of existing code.

- **Explanation and Communication:**
  Capturing and communicating system content as with an executive information system.

- **Decision Modeling:**
  Understanding and resolving design decisions and rationales.

- **Education and Training:**
  Training analysts and end users.

The difference to generalized knowledge representation projects that attempt to provide a basis for modeling encyclopedic knowledge is that domain modeling explicitly acknowledges that representations are designed for particular purposes. These purposes - the operational goals - inherently bias any particular solution and dictate the contents and the final form of the domain model.

The domain model designer, i.e. the domain analyst or the domain engineer has to obey these operational goals in his or her task of determining what knowledge about the application domain to represent, how to organize, and how to express it. Depending on that decision, the forms domain models can have vary to a large degree. A taxonomy, for example, which can be characterized as a definitional model, only shows what is in the domain and how it is organized. Knowledge representation models like the LaSSIE environment of Devanbu et al (Devanbu, Brachman, and Selfridge 1990) or the KITSS system of Kelly et al (Kelly 1991) provide semantics and some explanatory capabilities by semantic retrieval. Domain-specific languages like addressed in Neighbors DRACO system (Neighbors 1984),

when expressed as formal grammars supported by parsers, are models that may support direct translation of software specification into executable code.

There are also models that provide information on how to build systems for the domain. These may be in the form of standards, guidelines, templates, or interface definitions. Functional models describe how systems work using representations such as dataflow diagrams or program description languages. An examples for that is described in (Setliff 1991). Structural models describe how systems in the domain are built.

The construction of the domain model is called the model instantiation process in which the domain knowledge is expressed by using a meta-model or modeling language. A meta-model or modeling language is defined in the following way (Iscoe, Liu and Tam 1991):

"A language or representation structure used to specify the knowledge about a given domain. Formality and executability allow for reasoning and inference to support the operational goals."

Knowledge representation and conceptual modeling languages play a prominent role in making the output of the domain analysis explicit. Taxonomic and object-oriented representations are widely used. Conventional software engineering representation schemes such as structured charts, dataflow diagrams, state charts, or pseudocode are used to encode implementation knowledge. Hypertext media permit flexible interactions between analysts and domain models. Typing and algebraic specifications have a role in formalizing domain semantics (Prieto-Diaz, and Arango 1991).

CHAPTER 5
DOMAIN ANALYSIS AS A CONCEPT OF GenSIF

In chapter two the GenSIF framework has been introduced, followed by an elaboration of the context for the utilization of domain models within this framework in chapter three. A broad general overview of domain analysis which has been identified as a process that intends to derive and maintain a domain model has been given in the previous chapter.

Domain analysis as a concept of GenSIF has only been introduced on a top level without any discussion of its content. Only the general idea behind the framework in which domain analysis is intended to take place has been discussed so far. The assumption was that domain analysis in GenSIF is the prerequisite for the derivation of an integration architecture and in addition it is a preparation for the single application projects.

This chapter is concerned with the elaboration of domain analysis as a concept of GenSIF, i.e. with the elaboration of a domain analysis variant, in which the specific needs of the GenSIF framework are reflected. However the elaboration focuses on the domain modeling task for the reason that a modeling formalism which is appropriate for that domain analysis variant has not been identified yet. At first it is discussed how the application domain has to be analysed with respect to the specific needs of the GenSIF framework which basically corresponds to what has been called information acquisition in (Prieto-Diaz, and Arango 1991). Based on that discussion the relevant analysis results that have to be made explicit in a domain model are identified, in terms of real world phenomena. This declares the kind of domain information that a modeling formalism for GenSIF must be able to handle. (In section 5.2 it is attempted to give a rough specification for the general domain model type required by GenSIF.)

The point of view in the discussion of domain analysis in this chapter is changed then from aspects regarding the initial design of the domain model to aspects

concerning what basically happens if the domain model has been designed. Some of these aspects are discussed in section 5.4.

5.1 How to Analyse the Application Domain

From what we have said so far, we already can infer that domain analysis within GenSIF is concerned with domains relatively of large dimension, such as for example "company", "factory", or "university". The purpose of the domain analysis in that context is to provide a standardized model of the application domain. This model is used to provide information for the decision concerning the fitting integration architecture and to prepare the requirements analysis of the single application projects. That information should be made readily available in form of a domain model.

In contrast to other domain analysis approaches that incorporate a conceptual analysis as well as an constructive analysis, GenSIF, at the current state, does not require more than the former. It even is a fundamental principle of GenSIF that the domain analysis may not be oriented towards an "implementation" in a computer based system. The application domain is analysed from a general point of view, without any kind of application system in mind. It is viewed as what it is, namely a part of the real world that itself is embedded in other, larger parts of the real world.

In contrast to other kinds of real world analysis, domain analysis within GenSIF definitely does not deal with any raw quantitative data concerning the application domain. It stresses an "in breath" kind of conceptual analysis of the part of the real world that has been identified as the application domain. The acquired knowledge is not the one we typically find in Expert Systems. Instead of the analysis is concerned with shallow knowledge on a very wide basis.

The three underlying fundamental principles of that conceptual analysis are dis-

cussed in the following part.

Principle I:
Consideration of Three Relevant Real World Aspects

Viewing the application domain as a part of the real world implies the question of what are the particular aspects of the real world which should be analysed.

In the research seminar for systems integration at the New Jersey Institute of Technology, we have come to the decision that domain analysis within GenSIF has to be concerned with the following three aspects of the domain:

(1) Static Real World:

The term "static real world" is used here to refer to that aspect of the real world in which there is no time dimension, or in which the time dimension is unimportant at the moment. In principle we are talking about "objects" and "relations". If we take for example the application domain "university department", then the analysis result with respect to the static real world aspect of that example domain would be:

- there are students, professors, courses, etc.

- professors are instructors of courses

- students participate in courses

(2) Dynamic Real World:

With this term we refer to what is "happening" in the real world. Here the time dimension is of importance. We identify activities and processes. With respect to the example application domain "university department" some analysis results would be:

- at beginning of each semester students register for courses

- until December 15th in the fall semester students apply for graduation in the spring semester

(3) Communication that Takes Place in the Real World:

What is meant basically is relevant communication in its broad sense that happens in the real world. Hence this aspect focuses on information exchange. Typical analysis results could be:

- application forms for graduation are submitted by the students to the admissions office

- review process material is distributed by the office of the provost to the professors

Domain analysis within GenSIF goes beyond only an investigation of what has been called here the static real world due to the fact that we have to be complete in a way similar to requirements analysis for systems development.

The aspects which have been called "dynamic real world" and "communication that takes place in the real world" are also necessary to derive the integration architecture. It is quite plausible that for the definition of that integration architecture more than only a description of the static real world of the application domain is required. This can be claimed, since an integration architecture not only has to fit the static structure of the domain, but also the behavior of the elements in the domain and their communication. In general it can be said that the kind of domain analysis which is required in the GenSIF framework has to cover the static semantics as well as the dynamic semantics of the application domain.

static real world

Application

Domain

Model

dynamic real
world

communication in
the real world
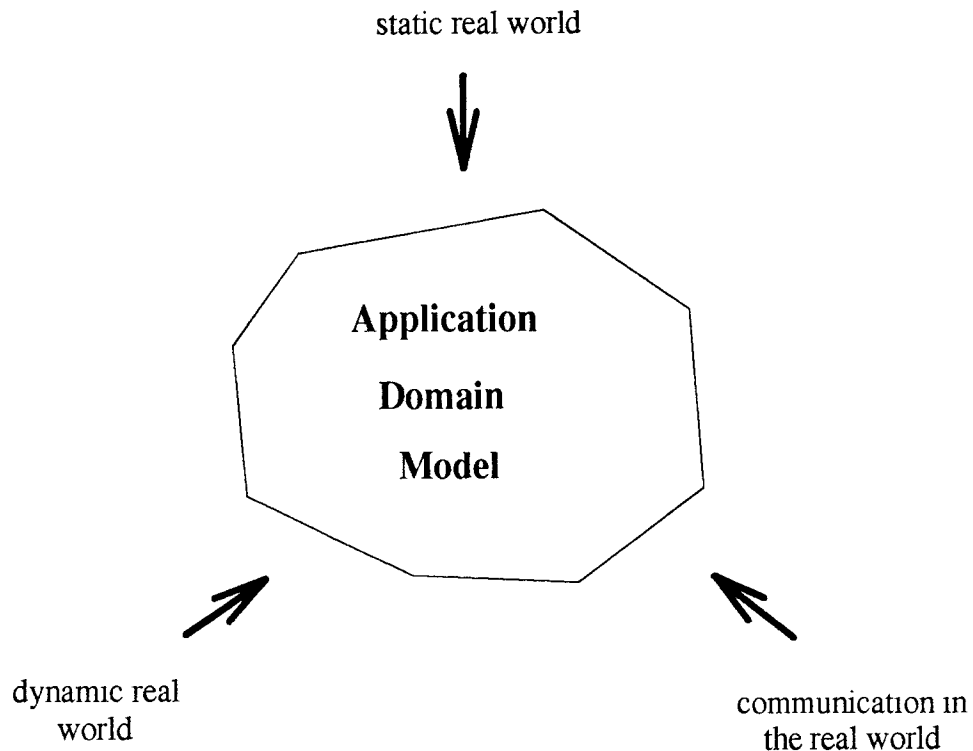
Figure 5.1: The three relevant aspects of the real world in domain analysis for GenSIF.

Principle II:
Consideration of Multiple Points of View

Another fundamental principle of domain analysis if it should meet the requirements of the GenSIF framework is that the analysis may not be restricted to only one specific point of view.

Theoretically speaking, if an application domain can be analysed with respect to one specific view V_i and there are j relevant, different points of view to look at the domain, then the domain analysis has to consider "all" these different points of view. With respect to the example application domain "university department" from above, points of view are for example:

- **View of a Student:**
  to look at the application domain from the point of view of a student

who is interested in academic programs, courses, his duties or for example
the consequences of an GPA lower than 3.3

- **View of a Secretary:**
  to look at the application domain from the point of view of an secretary
  of the department office, who probably needs to know the responsibilities
  of professors, performing organizational duties of the department

- **View of a Professor:**
  to look at the application domain from the point of view of a professor,
  who wants to know the procedure to be promoted from an assistant professor
  to an associate professor

All these examples correspond to points of view of elements which directly belong
to the domain but there are also examples for points of view from outside the
application domain at that domain like for example:

- **View of the Office of Sponsored Programs:**
  to look at the university department from the point of view of the office
  of sponsored programs which is concerned with the fund handling where
  the professors of the department are involved

- **View of the Admissions Office:**
  to look at the university department from the point of view of the admissions
  office which is concerned with the students of the department

- **View of the Office of the Provost:**
  to look at the university department from the point of view of the office
  of the provost which is, beside other concerns, involved in the promotion
  and tenure process

These different perceptions (views) of the application domain are important for the
discovery of the interdependencies between the items of the domain and to identify
roles. Furthermore it helps us to prepare a comprehensive domain model. It is
also important for the reason that the domain analysis should prepare the different
application projects which always have a different underlying point of view of the
application domain.

view of a secretary

view of
a student

University

Department

view
of a
professor

view of Admissions
Office
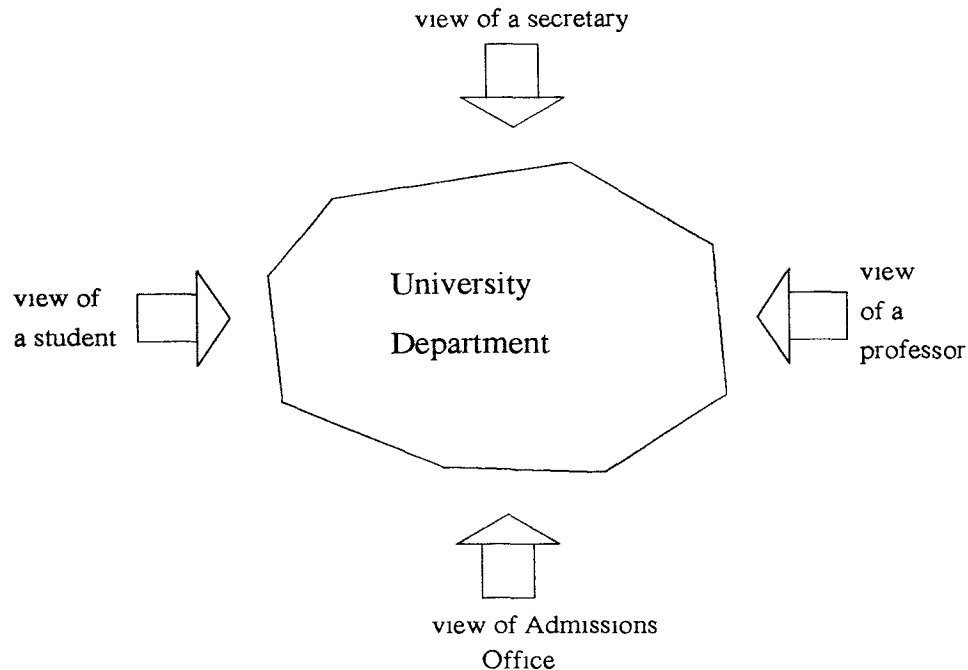
Figure 5.2: Different points of view at the application domain "university department".

Principle III:
Consideration of the "Interface" Between the Domain and the Domain Environment

In the introduction of this section it has been explained that within the GenSIF framework the application domain has to be viewed as a part of the real world that itself is embedded in another part of the real world. This implicitly has already defined a third principle for the domain analysis as it is required by GenSIF. Analysing the application domain is the central concern but this may not mean that the analysis can stop as soon as an imaginary boundary line is reached.

First of all, it has to be said that this imaginary boundary line between the application domain itself and the environment of the domain is very difficult to define, may be it is not possible at all. In (Prieto-Diaz, and Arango 1991) this has been called domain identification. But this is a research topic of its own, and is not investigated in this thesis.

The point that should be addressed here is that making the domain analysis

Environmental Object
Connected to the
Application Domain

Domain
Boundary → Application

Domain

Domain Environment
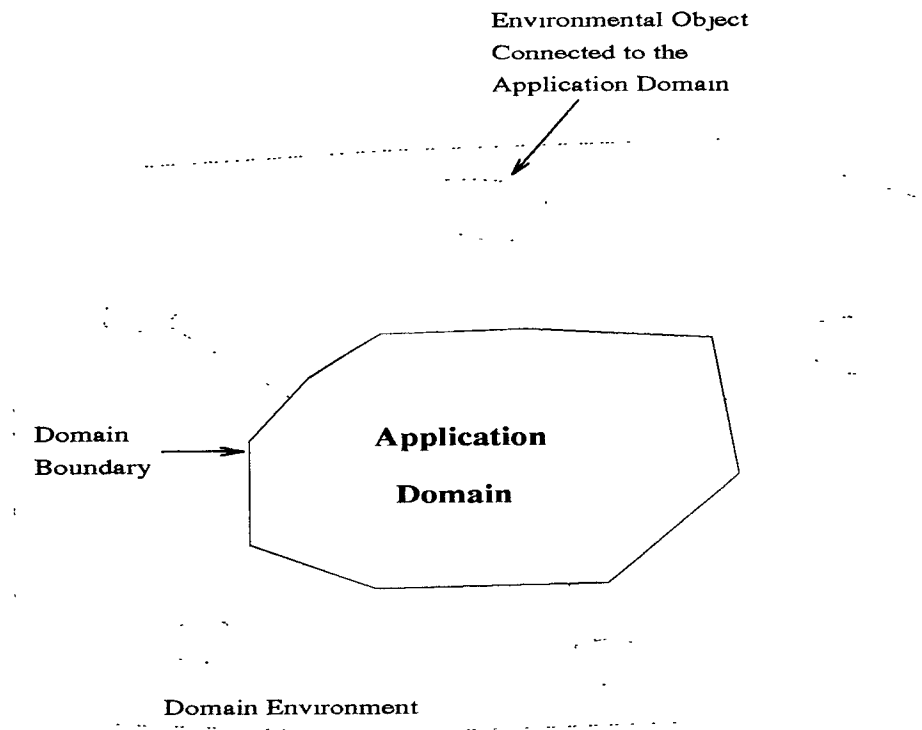
Figure 5.3: "Interface" between the domain and the domain environment.

in a strictly "closed world manner" is not sufficient. The scope and nature of the application domains which are analysed within GenSIF usually do not allow this, because this would lead to a lot of undiscovered semantics of things which belong to the domain. Or in other words, leaving the scope of the application domain is required if something in the domain is "tightly connected" with something outside the domain and for the discovery of its semantics needs an additional look at its connections to those "environmental things". Even if these "environmental things" are outside the current area of interest that has been identified as the application domain we would like to include them. This inclusion of the domain environment is not a transitive procedure, i.e. connections that "environmental things" might have to other "things" outside the scope of the application domain are definitely not considered any more. Furthermore "environmental things" should be identified as such by a qualitative different kind of description like in SADT sources and sinks.

An example for such an environmental object with respect to the "university department" application domain is the "travel expense report". (This example might be very specific for the New Jersey Institute of Technology where all the mentioned examples are derived from. However, it is assumed that it is not much different to the general case.) If a professor of an university department has received a fund from an institution, i.e. a company, the fund is partly administrated by an office that is called "office of sponsored programs". For every travel activity which is done for purposes that have been declared in the funding proposal, the professor has to make a travel expense report and submit this report to the office of sponsored programs, which than has to complete some "business as usual procedures". Restricting the analysis to a "closed world" analysis in this case would have the consequence, that the "hidden semantics" of the "travel expense report", namely that this form is requested by the "office of sponsored programs" and therefore is involved with the fund handling is not discovered.

5.2 A Domain Model "Specification"

In the previous section we have discussed how the application domain should be analyzed in domain analysis for GenSIF. This means we are now familiar with the variety of information that is considered in the analysis process. However what is missing yet at this point is something like a specification of the general type of domain model that is required by the GenSIF framework. To provide such a specification, although this is only possible on a rough level, is the intention of this section.

The specification is derived by providing first some key-words that identify the required usability of the domain model within the GenSIF framework. After that some fundamental general characteristics of the domain model are briefly discussed. Finally the important criteria of the phenomena description are provided at the end.

5.2.1 Required Usability of Domain Models for GenSIF

"Dictionary" for the Application Domain:

In a broader sense the domain model for GenSIF should be usable like a "dictionary" for the application domain. It defines a set of terms which are useful like a THESAURUS. For the example domain "university department" some of the terms could be: student, professor, course, laboratory, application form for graduation, academic program, ... etc. The meaning of each term in the real world is given by the connection between the real world and the model since it is assumed that the model has a formal semantic basis. This formal sematic basis is provided by the utilized modeling formalism. Once the domain model has been created, the user can obtain important information from the model that help to understand the application domain specific terminology.

"Knowledge Base" for the Application Domain:

The form of the domain model should be similar to a "knowledge base" for the domain. However, while knowledge of a real world domain may include heuristics and rules of inference as in artificial intelligence or expert-system knowledge bases, the knowledge acquired in the domain model is usually narrower in scope. It tends to include factual and deterministic knowledge. It includes no data knowledge and is without any portion of problem solving knowledge. The content of a domain model for GenSIF can be seen as "general knowledge", sometimes also called shallow knowledge.

5.2.2 General Characteristics of the Required Domain Model Type

As the next step, some more identifying general characteristics for the required type of domain model can be provided where the underlying point of view is not the usability of the model.

"Mental" Model:

A fundamental hypothesis of cognitive science (Psychology, Linguistics, Philosophy, Computer Science) is *"that people understand the world by building mental models (Sowa 1984)"*. If we accept this hypothesis and take into account that our domain model is based on the domain analyst's perception and understanding of the real world, than we might call it a model that reflects the domain analyst's personal mental model. This identifies one critical aspect that we have to consider within this context. The point is that the domain model has to be regarded as a subjective model that depends on the model designer's personal perception and understanding of the real world.

"Exclusive" Real World Model:

In contrast to other approaches the domain model as product of domain analysis in GenSIF is not targeted towards system design. It only is concerned with the part of the real world that has been identified as the relevant application domain.

5.2.3 Important Criteria for the Phenomena Description

Human Oriented Phenomena Description:

As discussed in chapter three, the users of the domain model within GenSIF are system engineers and system developers. Although it has not been elaborated in which way the domain model is exactly utilized by these users, it can be infered that the domain model within GenSIF has to be a human oriented model in the sense that what the model describes must primarily be understandable for humans. In contrast to that, a simulation model must primarily be "understandable" by a computer. Nevertheless, it is important that the entire description is well structured and organized and the description is not ambiguous.

Usability:

* "Dictionary" for application domain
* "Knowledge Base" for application domain

Domain Model
for
GenSIF framework

Phenomena Description:

* human oriented
* abstract model

General Characteristics:

* real world model
* "mental" model
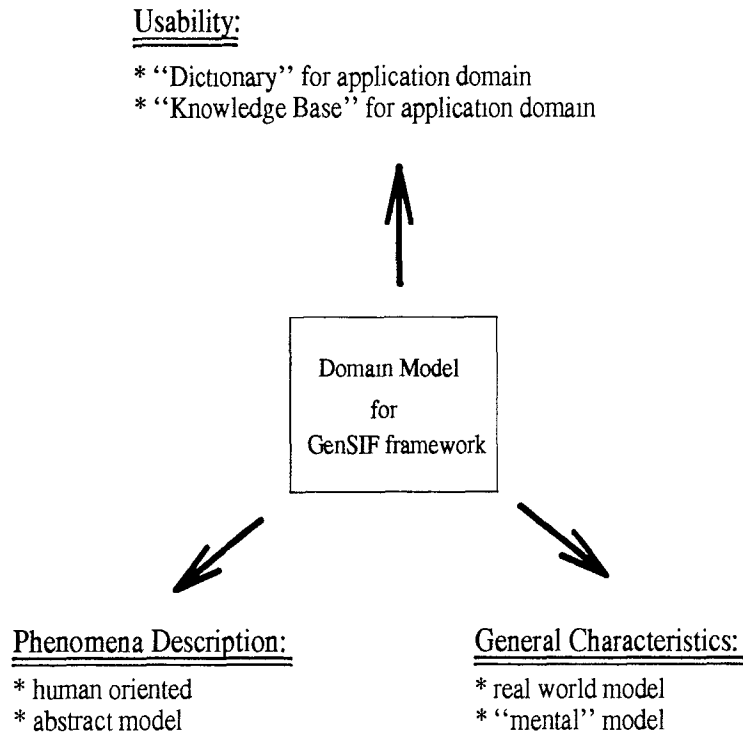
Figure 5.4: Rough "specification" of the required domain model type for GenSIF

Abstract Phenomena Description:

A domain model within GenSIF is an abstract description of the application domain, in which relevant aspects are described and other irrelevant aspects are omitted. It is like the abstract description of a real building, which might describe the shapes of the main components, their relative sizes, and their relative positions and orientations, but it may omit to describe the material from which the walls are built, the water, electrical and other services, the parts of the building that are underground, and the internal structures of the building. Jackson (Jackson 1983) has described this kind of selectivity as inevitable in any description, since "... *the only complete description of reality is the reality itself*".
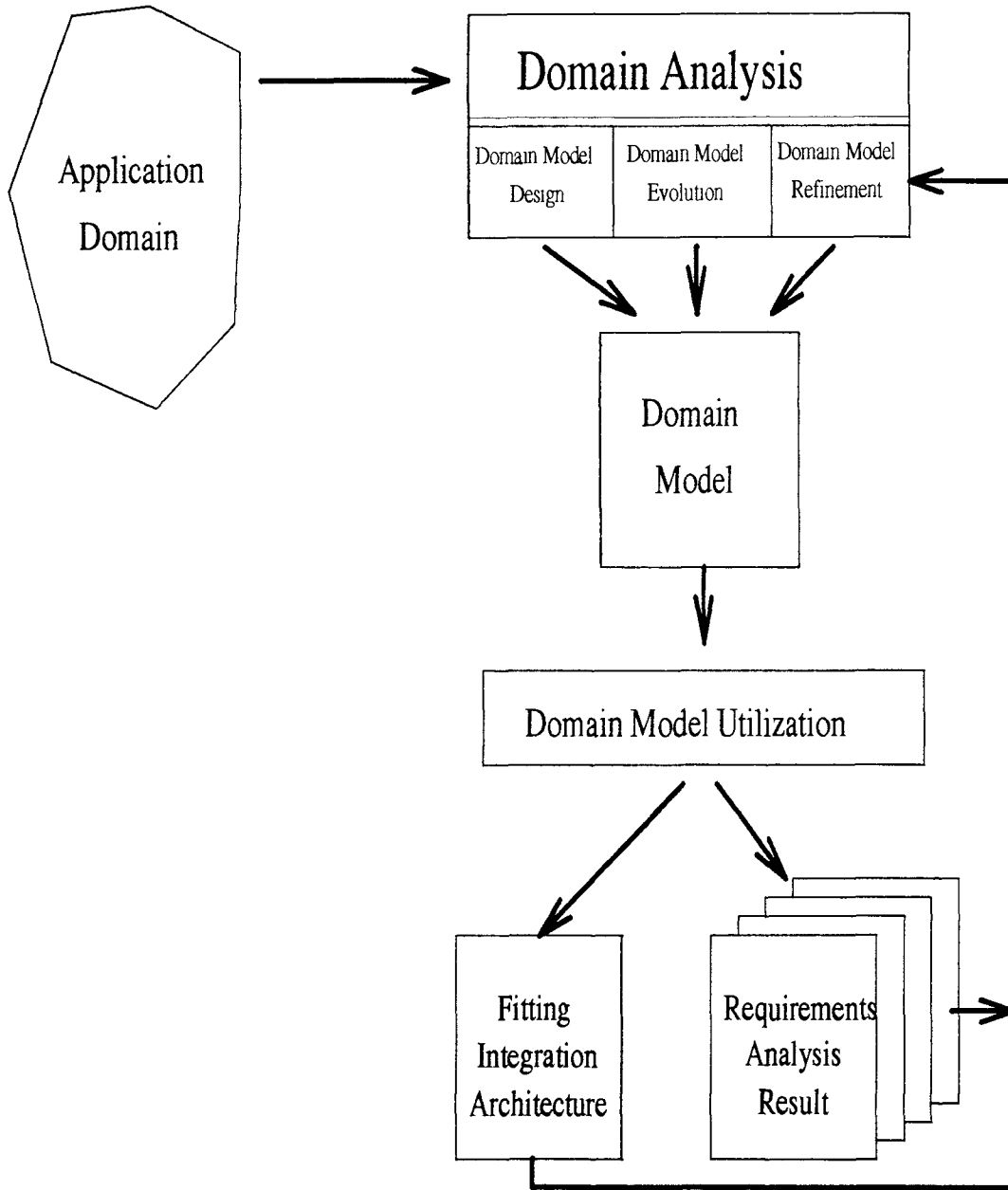
Figure 5.5: "Infinity" of the domain analysis process within GenSIF.

5.3 "Infinity" of the Domain Analysis Process within GenSIF

Domain analysis as a concept of GenSIF is not a process that is completed as soon as a domain model has been derived. In fact it is an ongoing process that only is finished under the condition that there is no longer any interest in the application domain. Therefore it can be concluded that domain analysis as a concept of GenSIF is in some sense an infinite process.

Why is it an infinite process can be explained by the fact that from GenSIF's point of view systems engineering and systems development for large integrated systems is not something that takes places only once or ends in a determined time interval. Once the domain model has been designed and the integration architecture has been selected or developed, some application projects will be identified and developed. However, there might be other projects in the future for which the need has already been identified or which will emerge just in the future. Since application domains are dynamic and change over time, these changes must be considered and the domain model must be updated, if these future application development projects also should have the domain model as input for their requirements analysis.

Some changes within the application domain might even require changes in the integration architecture, because it is an essential requirement of GenSIF that the integration architecture fits to the domain.

This "updating" of the domain model, i.e. keeping it an accurate description of the application domain, is also part of the domain analysis process. It directly corresponds to what has been called "Domain Model Evolution" in (Prieto-Diaz, and Arango 1991). Another aspect of domain analysis that follows the initial design of the domain model is what can be called the refinement of relevant domain information. Refinement includes mainly the insertion of more specific domain information that has been acquired by the requirements analysis of application development projects. This is done for the reason that every additional, relevant domain infor-

Described
Phenomena

Domain

Model

Description

Selected
Phenomena

Selection

Perceived
Part of Real World

Perception
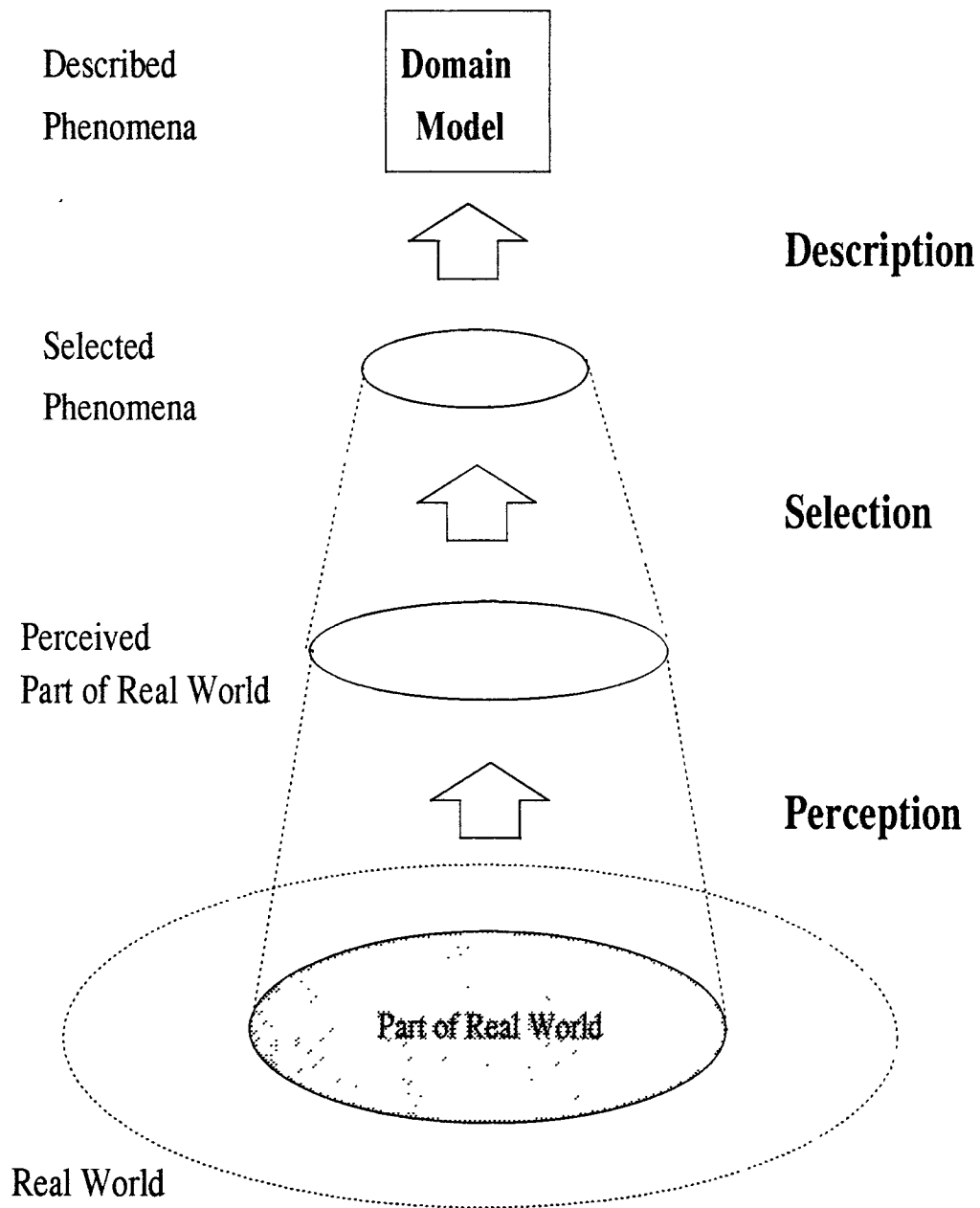
Part of Real World

Real World

Figure 5.6: Selecting domain phenomena as domain model content.

mation should also be made readily available to other projects, together with what already has been made explicit by the design of a domain model. Hence, the more general and generic original domain model becomes a more and more specific and specialized model of the application domain during this refinement process.

5.4 Domain Phenomena to Select as Domain Model Contents

In section 5.1 it has been explored how the application domain should be analysed if the specific requirements of the GenSIF framework are reflected in the domain analysis process. The main conclusion regarding that investigation was that in the information acquisition step of the domain analysis process, information is derived by viewing the application domain simply as a part of the real world and performing a conceptional analysis of this part of real world with respect to the following three principles:

1. consideration of three relevant real world aspects,

2. consideration of multiple points of view,

3. consideration of the "interface" between the domain and the domain environment.

This basically can be understood as guidance how the application domain should be perceived in that kind of analysis (It is assumed that the real world, i.e. the application domain, extends beyond that perceived part.). The purpose of this section is to discuss the relevant domain phenomena that should be selected during the analysis of the perceived application domain. The description of all these selected domain phenomena basically forms the domain model. In this "description-process" the selected phenomena are mapped into the modeling primitives provided by the applied modeling formalism (Figure 5.6).

5.4.1 Relevant Domain Phenomena

We in the research group for systems integration at the New Jersey Institute of Technology (spring term 1992) have identified and broadly classified in a pragmatic manner the phenomena that should be described in a domain model for GenSIF. A good help to avoid confusion with the terms used in object oriented modeling approaches is the following differentiation criteria: The terms given below do only refer to "what" we would like to have described. However, "how" it should be described is not considered here at all.

(As throughout this whole thesis, the presented examples for relevant phenomena are taken from the application domain "university department".)

(1) Domain Object-Classes

Before it can be defined what is meant with a domain object-class, first it is necessary to define the term domain object. The given definition has been adapted from (Embley, Kurtz, and Woodfield 1992) where it is given within the context of Object-Oriented Systems Analysis.

□ Definition of Domain Object:

An object is a person, place, or thing. It may be physical or conceptual. The idea is that an object is a single entity or notion. Each object is a unique individual. An object may be related to or made up of other objects, but each object is unique. If the object belongs to the application domain, then we call it a domain object.

□ Examples for Domain Objects:

Examples for physical domain objects:

- the student `David with Student-ID-Number 017-90-4011`
- the workstation `newark3 with Inventory Number 12034-1117-200`
- the laboratory `Software Engineering Laboratory`
- the data storage `File of Graduating Students`

Examples for conceptual domain objects:

- the course `CIS 631`

- the academic program `Master of Science in Computer Science`

- the final exam `CIS 631 at 05/02/1992`

□ **Definition of Domain Object-Class:**

A set of domain objects that belong together for some logical reason is a domain object-class. Which domain objects belong together depends on the perception of the domain analyst. Each object-class has a name that is generic and denotes any member of the object-class.

□ **Examples for Domain Object-Classes:**

- `PERSON`

- `STUDENT`

- `PROFESSOR`

- `COURSE`

- `LABORATORY`

- `ROOM`

The following relevant phenomena are descibed on an object basis. By the term "classified objects" which I use several times throughout their discussion I refer to objects which are the members of an object-class. However in the final domain model we want to have a description on an object-class level.

(2) Relationships between Domain Objects

□ **Definition of Relationship:**

A relationship is a logical connection between objects that associates one object with other objects.

□ **Examples for Relationships:**

- thesis advisors, which are professors, `advise` students

- courses are `taken` by students

- courses are `taught` by professors

(3) Roles of Domain Objects

□ Definition of Role:

A role is a specific point of view on domain objects that belong to a specific domain object-class. More concrete a role can be regarded as a mission, assignment, job, or purpose that belongs to the objects of a specific domain object-class. Roles can be differentiated according to several different criteria, like level of commitment towards the role (mandatory, optional) or frequency of the role (one time, repetitive, permanent).

□ Examples for Roles of Domain Objects:

Relevant roles for the objects in the domain object-class `REGISTERED STUDENT` are:

- `research assistant`

- `course participant`

- `Master's Thesis writer`

- `participant in academic program`

Relevant roles of objects in the domain object-class `PROFESSOR` are:

- `course instructor`

- `thesis advisor`

- `researcher`

- `promotion candidate`

(4) Activities of Domain Objects

□ Definition of Activity:

Something that is done is called an activity. Objects are the processors of activities. Hence, certain habitual activities can be identified as those activities that are performed by certain objects classified in an object-class.

(We would get the behavior pattern of the objects classified in an object-class if we also would consider the sequence of the activities. See section 5.4.2)

□ Examples for Activities of Domain Objects:

Activities of the objects in the domain object-class REGISTERED STUDENT are:

- apply for graduation
- register for courses
- make Master's Thesis proposal

Activities for the objects in the domain object-class PROFESSOR are:

- make funding proposal
- process annual summary form
- make travel expense report

(5) Communication between Objects

□ Definition of Communication:

The exchange of information between objects is regarded as communication between objects. A differentiation between the participating partners (environmental object or classified object) is not made.

□ Examples for Communication between Objects:

- review process material is exchanged between the OFFICE OF THE PROVOST and PROFESSORS which is implied by the promotion and tenure process
- the travel expense report is exchanged between PROFESSORS and the OFFICE OF SPONSORED PROGRAMS

(6) Environmental Objects

□ Definition of Environmental Object:

It is referred to the definition for the term "Domain Object" given above. In contrast to that, if an object is of relevance but does clearly not belong to the application domain any more it, is referred to as an environmental object. Please see also section 5.1, principle III, for an explanation regarding the question when such an environmental object is of relevance.

□ Examples for Environmental Objects:

- `OFFICE OF SPONSORED PROGRAMS`
- `OFFICE OF GRADUATE STUDIES`
- `OFFICE OF THE PROVOST`

(7) Activities of Environmental Objects

□ Definition of Activity of an Environmental Object:

In contrast to activities of classified domain objects, here the activity is performed by what has been defined as an environmental object.

□ Examples for Activities of Environmental Objects:

In the following list the environmental object which is the processor of the environmental activity is given in parenthesis.

- `perform promotion and tenure process (OFFICE OF THE PROVOST)`
- `close course (OFFICE OF THE REGISTRAR)`
- `hire students (COMPANY)`
- `establish fund (COMPANY)`

(8) Time Frames

□ Definition of Time Frame:

As the name suggests a time frame is a certain period of time which is associated

with (a) certain (activity) activities. Since a time frame identifies the time interval in which (an activity is) activities are performed in the domain, it can also be regarded as a temporal constraint for the (activity) activities.

□ **Examples for Time Frames:**

- academic year
- term
- graduation application period
- registration period

5.4.2 Currently Unconsidered Domain Phenomena

At the current status of our research regarding domain analysis within the GenSIF framework, we have limited the contents of the domain model, to the elements listed in section 5.4.1. However there are other elements which we have regarded as currently not important for our domain model. These notions could become relevant if we advance in our research.

(1) Behavioral Patterns

What we mean with behavioral pattern is the identification of the sequence in which activities are performed. This could also include that one activity is identified as a subactivity of another activity. Furthermore this could mean that several activities which are oriented towards the same global goal are identified as something that might be called a process.

(2) States and State Transformations

Although we are interested in the dynamics of the real world, currently we do not consider the notions of state and state transformations like it is done in Petri-Nets.

At the actual status of our research we want to consider dynamics on a more general level.

(3) Internal Events

An internal event is an instantaneous happening in the real world that has an impact on one or more phenomena of the application domain. An event can trigger certain additional activities, suppress certain activities, or change the status of objects. At the moment we rather see the consideration of such internal events as a task of the requirements analysis for the specific projects within the application domain.

(4) Constraints and Rules

Currently we see the consideration of constraints and rules as a task of the requirements analysis for the specific projects within the application domain.

(5) Quantities and Heuristics

Since domain analysis for GenSIF at the current status stresses an abstract conceptual kind of analysis, quantities and heuristics do not have first priority. However as selected quantities and heuristics they are of interest in general.

(6) Pre- and Postconditions of Activities

This simply regards to the conditions that must be satisfied before an activity can be performed or after it has been completed respectively. Since we motivate an abstract kind of analysis we do not consider these phenomena.

5.5 Comparison to Other Domain Modeling Approaches

In the preceding sections of this chapter a domain modeling approach has been introduced, which is based on the specific needs of the GenSIF framework. It is

the intention of this section to compare this approach to other domain modeling approaches on a general level. The focal point of the comparison is on the contents and form of the domain model.

In contrast to most other approaches, the GenSIF specific domain modeling approach strives to build a much more comprehensive and differentiated model of the application domain. The information provided by that model is not restricted to only the objects in the domain, and their relationships to each other as it is typical for domain modeling for reuse purposes. Domain modeling within the Gen-SIF framework strives to capture more than this. As elaborated in the succeeding sections it also is concerned with the dynamics, and the communication within an application domain. Furthermore, in contrast to other approaches, it is the intention to describe the domain from as many different perspectives as necessary. Another difference is that, if necessary, information regarding the domain environment are captured in the domain model as well.

As an intermediate conclusion it can be said that a domain model within GenSIF is concerned with a larger variety of phenomena than other approaches.

In (Prieto-Diaz, and Arango 1991) two facets of domain analysis are proposed which are called: *conceptual analysis* and *constructive analysis*. This is the typical reuse oriented domain analysis view where the domain model includes in addition to the results of the conceptual analysis of the application domain, information obtained by a constructive analysis as well. For instance in (Prieto-Diaz, and Arango 1991) it is claimed that "a model of a domain should include information on at least three aspects of a problem domain":

- concepts to enable the specification of systems in the domain
- plans describing how to map specifications into code
- rationales for the specification concepts, their relations, and their relation to the implementation plans

As a significant difference the domain modeling approach within the GenSIF framework is not concerned with any information regarding a constructive analysis at all. It even is one of the basic principles of the way domain analysis should be performed for GenSIF that the application domain should be perceived as "neutral" as possible. With "neutral" we mean that the analysis may not be targeted towards any software systems solution.

There are some domain modeling approaches in which the domain model contents include the domain modeling history (Lubars 1988) as well. Basically the domain modeling history is the recording of events that affect the state of the domain model. Currently the domain modeling approach within GenSIF does not include such a component. Although it would be nice to have this component, at the actual state of our research it would be more of a luxury feature than a necessity.

CHAPTER 6
AN APPROACH TO DOMAIN MODELING
FOR GenSIF

This chapter introduces an intuitive and pragmatic approach to a domain modeling notation which has been developed based on what has been described in section 5.4 as the domain phenomena to select as domain model content. It should be pointed out that there is no formal proof that the introduced notation is sound and complete, and that we do not claim that it is the ultimate and only alternative to the task addressed.

It has to be understood as the documentation of an experiment. This experiment was carried out to gain more insights into a domain modeling notation for a domain analysis variant in which the particular needs of the GenSIF framework are reflected.

In the first section the main idea behind the approach is introduced. In order to enable the continuation of the introduction of the approach by discussing concrete modeling examples, section 6.3 is concerned with the specification of modeling primitives. Section 6.4 applies the primitives to model the application domain "university department". Finally, in the last section, the contributions of this design experiment for the GenSIF project are analysed from the authors point of view.

6.1 The Philosophical Basis

The underlying basic opinion for the experiment to design an approach to a specific domain modeling notation for GenSIF is that such a notation contributes best to the goal if it provides a rich set of adequate modeling primitives. The concrete modeling primitives, introduced in the next section, were determined by looking at the real world phenomena that have to be described in domain analysis for GenSIF (see section 5.4).

This stands directly in contrast to some other opinions that argue for a limited set of modeling primitives to gain a high processability of the model and to simplify the model design process. The study of the relevant research literature shows that this currently is an open issue that has lead to the separation of two main "modeling schools". There is the "modeling school" that favoures a rich set of adequate modeling primitives that allows to map relevant phenomena very directly into the formalism. Some representatives are for example Robert Balzer's GIST specification language (Balzer 1981), and the ERAE data model for requirements analysis developed by Eric Dubois et al (Dubois, Hagelstein, Lahou, Ponsaert, and Rifaut 1986; Dubois, Hagelstein, Lahou, Rifaut, and Williams 1986). On the other hand side there is the "modeling school" that claims a limited set of primitives with which the model designer can describe the relevant phenomena. The approaches that can be regarded as representatives of that school typically do not facilitate a specific modeling formalism. Instead they map everything into the primitives typically supplied by object-oriented languages or knowledge representation languages. Booch's object-oriented Ada design method (Booch 1987) is one of the representatives of that "modeling school".

Another underlying opinion is that with respect to the tasks that should be facilitated by the required domain model type, modeling behavior by means of operational semantic of the modeling formalism is not a good solution. Although operational semantic provides capabilities for model validation by model execution, the study of modeling languages like GIST (Balzer 1981; Goldman, and Wile 1980) has shown that this implies in some sense an unnatural way of modeling behavior. Unnatural, because the real world behavior has to be encoded in data base like atomic transactions, as for example:

- create object, create relation
- destroy object, destroy relation
- insert object, insert relation

For that reason, our approach to a domain modeling notation can be called an unified approach, since everything is based on the same descriptive paradigm, with no operational semantic at all. Hence the approach presented in the next sections reminds to some degree on the requirements modeling language RML which has been developed by Sol Greenspan (Greenspan 1984). It also has something in common with the ERAE model invented by Eric Dubois et al (Dubois, Hagelstein, Lahou, Ponsaert, and Rifaut 1986; Dubois, Hagelstein, Lahou, Rifaut, Williams 1986)

Despite some similarities, the presented notation is not supposed to be a variation of the semantic network formalism or even a data model. The spirit of the notation is much closer to the spirit of object-oriented modeling techniques such as proposed in (Embley, Kurtz, and Woodfield 1992) and (Rumbaugh, Blaha, Premerlani, Eddy, and Lorenson 1991). As stated in the introduction, our approach should be understood as a specific notation targeted towards the specific information we would like to formalize in a domain analysis within GenSIF. With our proposed notation we are currently able to reach a semiformal kind of description of what is important within our domain analysis. A more elaborated discussion of the contributions of the design experiment to the GenSIF project is provided in section 6.5. The shortcomings of the notation are discussed in subsection 8.2.2.

6.2 The General Approach

Using the proposed notation leads to a representation structure that could be compared to a "semantic network" of building blocks, i.e. particular modeling primitives provided by the notation. The provided primitives can be differentiated in two groups. One group is used to form the nodes of the network and, therefore, can be called node-primitives. The other group is used to form the links between the nodes and can be called link-primitives or just simply connectors. Similar to

semantic networks, the node-primitives represent concepts and the link-primitives represent relationships between the concepts represented by their corresponding node-primitive.

The provided node-primitives are derived from the phenomena which are relevant for domain analysis within GenSIF (see sections 5.1 and 5.4). Node-primitives are basically generalized constructs to describe the set of interesting phenomena in a way that allows to take the specific type of domain model, required by GenSIF, into account (see section 5.2).

The notation provides two types of link-primitives:

- **completely predefined relationship connectors:**
  Applicable to accomplish a connection between two specific types of predefined node-primitives with a predefined specific semantic. This means the identifier of the relationship connector as well as the semantic of the connector is completely predefined.

- **partially predefined relationship connectors:**
  Applicable to accomplish a connection between two specific predefined node-primitives with a partially specific semantic. The semantic is only partially predefined since the complete semantic of the relationship is only given when the primitive has been associated by the model designer with a proper word (identifier) denoting the relationship-name.

To develop a domain model means to select from the predefined primitives the fitting ones and to use them to describe an abstraction of the real world. This restricts in some sense the freedom of expression for the model builder. However, at the same time it opens the possibility to guide the model design process towards a final model that is based on a standardized set of well defined constructs. This allows one to handle domain analysis as an engineering process and to map the model, expressed in well understood primitives, to a (partially) machine interpretable formalism.

6.3 The Modeling Primitives

The particular modeling primitives provided by the notation have been derived by looking at the phenomena that are relevant for a domain model for GenSIF (see section 5.4). Hence the notation allows to map each relevant phenomena to its own primitive.

The complete set of primitives is informally introduced in this section. According to what has been said in section 6.2, the primitives can be differentiated in those that correspond to nodes (node-primitives) and others that correspond to links (link-primitives) in the final diagrammatic structure. Furthermore link-primitives can be differentiated in those that are completely predefined (graphically depicted as single solid line arrows) and others that are partially predefined. Three different partially predefined link-primitives are provided:

- **Object-Class-Relationship Link-Primitive:**
  graphically depicted as arrow consisting of three solid lines

- **Role-Relationship Link-Primitive:**
  graphically depicted as double solid line arrow

- **Environmental-Participation-Relationship Link-Primitive:**
  graphically depicted as single dotted line arrow

In the following, I introduce these primitives by describing each node-primitive together with the link-primitives that are applicable to accomplish a connection from that primitive to a particular other primitive. As already explained, an association between primitives represents a relationship between two real world concepts which are represented by node-primitives.

Under the headline "Connectivity" I discuss the issue if the number of connections a node-primitive can have to other node-primitives is restricted. I provide this discussion for every type of node-primitive and link-primitive.

It should be pointed out that the names used for the predefined relationships have been chosen from a practical point of view. Although that choice was made with general applicability in mind the overall objective was to accomplish a development state that enables to present the spirit of the approach by some concrete modeling examples. An overview of all node primitives and their links can be found in the appendix.

(1) The Node-Primitive "(Domain) Object-Class"

□ **Definition:**

An object-class corresponds to a collection of abstract or concrete domain objects that, presumably are grouped together because some uniform conditions hold over all of them.

□ **Diagrammatic Symbol:**

□ **Applicable Link-Primitives:**

- *"category_of":*

An object-class can be connected to another object-class by this predefined link-primitive to represent that one object-class (tail of arrow) is a specific category of objects (a subset), contained in the other, more general, object-class (head of arrow).

- *Object-Class-Relationship:*

This link-primitive can be used to connect one object-class with another object-class or a role to represent a particular relationship between both concepts. The model designer has to denote the relationship by a proper name. In order to be able

to differentiate this link-primitive which is only partially predefined from others, the graphical representation for the object-class-relationship is an arrow consisting of three solid lines. Furthermore the relationship name should be written in capital letters.

General Diagrammatic Symbol of the Object-Class-Relationship Link-Primitive:
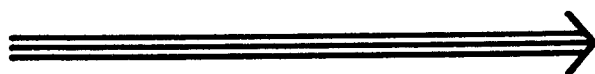
□ **Connectivity:**

A primitive of type object-class may only be connected to exactly one other primitive of type object-class by the connector *category_of*. In contrast to that there is no restriction for the numbers of connections that can be established by the link-primitive *Object-Class-Relationship*.

(2) The Node-Primitive "Role"

□ **Definition:**

A role is a specific view on the objects contained in a particular object-class. However with view we do not mean the same what is regarded in the database area as a view. In our context a view is oriented towards operationality. Hence a role is a medium that allows to group the specific set of activities that are performed by the objects of a particular object-class. Furthermore it helps to implement different user views on the application domain. The role name denotes a specific mission, job, purpose, duty or assignment for which the following holds: 1. the objects in the connected object-classes are the owners of that mission, job, purpose, duty, or assignment, and 2. the activities connected to the role specify the operational side of that mission, job, purpose, duty or, assignment. Since a role in our context need not necessarily to have an operational side (passive role), they are not required to

have activities which are connected to them. While a role can represent a mission, job, purpose, duty, or assignment which can have several different owners, a role can be connected to several different object-classes.

□ Comments:

The notion of role can also be found in other modeling approaches as well. Here usually roles correspond to the relationships between the object-classes. It is an important difference that in our approach a role does not represent a relationship between concepts. Instead it is a concept of its own.

□ Diagrammatic Symbol:
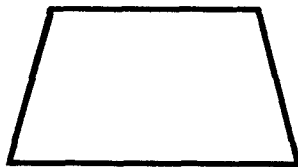
□ Applicable Link-Primitives:

- *"mandatory_one_time_role_of"* abbreviated *"mot_role_of"*:

This predefined link-primitive can be used to connect a role (tail of arrow) with a relevant object-class (head of arrow) to represent that the role is mandatory for exactly one time with respect to the owner objects contained in the connected object-class.

- *"mandatory_repetitive_role_of"* abbreviated *"mr_role_of"*:

This predefined link-primitive can be used to connect a role (tail of arrow) with a relevant object-class (head of arrow) to represent that the role is mandatory for more than only one time with respect to the owner objects contained in the connected object-class.

- *"mandatory_permanent_role_of"* abbreviated *"mp_role_of"*:

This predefined link-primitive can be used to connect a role (tail of arrow) with

a relevant object-class (head of arrow) to represent that the role is permanently mandatory with respect to the owner objects contained in the connected object-class.

- *"optional_one_time_role_of"* abbreviated *"oot_role_of":*

This predefined link-primitive can be used to connect a role (tail of arrow) with a relevant object-class (head of arrow) to represent that the role is not mandatory with respect to the connected object-class. Furthermore it specifies that if an object contained in the connected object-class is owner of that role than the ownership cannot be repeated.

- *"optional_repetitive_role_of"* abbreviated *"or_role_of":*

This predefined link-primitive can be used to connect a role (tail of arrow) with a relevant object-class (head of arrow) to represent that the role is not mandatory with respect to the connected object-class. Furthermore it specifies that if an object contained in the connected object-class is owner of that role than the ownership can be repeated.

- *"optional_permanent_role_of"* abbreviated *"op_role_of":*

This predefined link-primitive can be used to connect a role (tail of arrow) with a relevant object-class (head of arrow) to represent that the role is not mandatory with respect to the connected object-class. Furthermore it specifies that if an object contained in the connected object-class is owner of that role than the ownership is permanent.

- *Role-Relationship:*

This partially predefined link-primitive can be used to connect one role with another role or an object-class to represent a particular relationship between both concepts. The model designer has to "fill in" the name of the relationship similar how it is done in the design of Entity-Relationship Diagrams. However in contrast to relationships in ER-Diagrams role-relationships are graphically depicted as arrows. In order to be

able to differentiate this only partially predefined link-primitive from others in the final model their arrow consists of double lines. Furthermore the relationship name should be written in capital letters since the names of all predefined link-primitives are written in small letters.

General Diagrammatic Symbol of the Role-Relationship Link-Primitive:
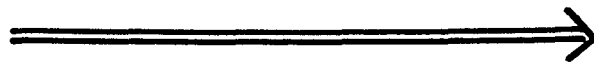
☐ **Connectivity:**

A primitive of type role may only have exactly one connection to one and the same primitive of type object-class by one of the six different connectors *xxx_role_of*. However the number of connections to different primitives of type object-class is not restricted. There is no restriction with regards to the application of the link-primitive *Role-Relationship*.

(3) The Node-Primitive "Activity"

☐ **Definition:**

The activity primitive is used to represent habitual activities. An activity either is connected to a role or directly to an object-class. In both cases by this primitive the operational side of the objects which are contained in the indirectly (with a role as intermediate primitive) or directly connected object-class can be specified. In our context a role has been defined as mission, job, purpose, assignment, or duty. Since activities can belong to different missions, jobs, purposes, assignments, or duties, it is allowed to connect an activity to several different roles even if the role owner object-class is different. Furthermore an activity can directly be connected to several different object-classes as well. For an activity it is not necessary to be connected to a corresponding time frame primitive. Here the model designer has

to decide whether or not to consider temporal aspects with regards to the activity.

□ **Diagrammatic Symbol:**

□ **Applicable Link-Primitives:**

- *"mandatory_one_time_activity_of"* abbreviated *"mot_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with either a role or an object-class (head of arrow). The semantics of this connector is that the activity is mandatory for exactly one time with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to.

- *"mandatory_repetitive_activity_of"* abbreviated *"mr_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with either a role or an object-class (head of arrow). The semantics of this connector is that the activity is mandatory for more than only one time with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to.

- *"mandatory_permanent_activity_of"* abbreviated *"mp_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with either a role or an object-class (head of arrow). The semantics of this connector is that the activity is mandatory permanently with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to.

- *"optional_one_time_activity_of"* abbreviated *"oot_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with

either a role or an object-class (head of arrow). The semantics of this connector is that the activity is not mandatory with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to. Furthermore it is predefined that if the activity is performed once than it cannot be repeated.

- *"optional_repetitive_activity_of"* abbreviated *"or_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with either a role or an object-class (head of arrow). The semantics of this connector is that the activity is not mandatory with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to. Furthermore it is predefined that the activity can be repeated.

- *"optional_permanent_activity_of"* abbreviated *"op_activity_of"*:

This predefined link-primitive can be used to connect an activity (tail of arrow) with either a role or an object-class (head of arrow). The semantics of this connector is that the activity is not mandatory with respect to the objects in the object-class it is directly or indirectly (role as intermediate primitive) referred to. Furthermore it is predefined that the activity is performed permanently.

- *"scheduled_in"*:

By this predefined link-primitive temporal aspects with regards to performed activities can be considered. It can be used to connect an activity (tail of arrow) with an primitive that is introduced later on and which is called time frame (head of arrow). The predefined semantics of this primitive is that the activity only is performed within the connected time frame but not outside that time frame. Hence a connection of an activity to a time frame by the application of this link-primitive is like the specification of a temporal constraint for the activity.

□ **Connectivity:**

A primitive of type activity may have only one connection to one and the same primitive of type role by one of the six different connectors *xxx_activity_of*. Like-

wise there may only be one connection between one particular primitive of type activity and one and the same primitive of type object-class. However the number of connections to different primitives of type role (no matter if the owner object-class is identical) and type object-class is not restricted. There is no restriction with regards to the link-primitive *scheduled_in*.

(4) The Node-Primitive "Communication Element"

□ **Definition:**

The primitive communication element is used to represent items that are exchanged between a sender and a receiver as part of an activity of the sender. All three components must be specified. Both, sender, and receiver, can be an environmental object, a role of an object-class, or an object-class itself.

□ **Diagrammatic Symbol:**

□ **Applicable Link-Primitives:**

- *"sent_by":*

This predefined link-primitive is used to connect the communication element (tail of arrow) with the sender (head of arrow) which either is an environmental object, a role of an object-class, or an object-class itself. The predefined semantics of this primitive is that the communication element is sent by the component it is connected to.

- *"received_by":*

This predefined link-primitive is used to connect the communication element (tail of arrow) with the receiver (head of arrow) which either is an environmental object,

a role of an object-class, or an object-class itself. The predefined semantics of this primitive is that the communication element is received by the component it is connected to.

- *"implied_by":*

This predefined link-primitive is used to connect the communication element (tail of arrow) with the activity of the sender (head of arrow). The predefined semantics of this primitive is that the connected activity implies the exchange of the communication element between the connected sender and receiver. Since the sender can be an environmental object the activity can be an environmental activity as well.
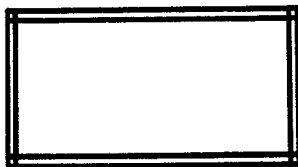
□ **Connectivity:**

The three different link-primitives introduced in the previous paragraph must be applied for each primitive of type communication element. However each of these connectors may only be used one time so that the total number of connections for primitives of type communication element is restricted to the number three.

(5) The Node-Primitive "Environmental Object"

□ **Definition:**

The modeling primitive environmental object is used to represent a concrete or abstract object which is outside the scope of the application domain. For more information regarding the consideration of environmental objects within domain analysis for GenSIF the reader is referred to section 5.1 principle III.

□ **Diagrammatic Symbol:**

□ Applicable Link-Primitive:

- Environmental-Participation-Relationship:

This partially predefined link-primitive is used to connect an environmental object (tail of arrow) with an domain activity (head of arrow). The general meaning of this link-primitive is that the environmental object is involved in the domain activity it is connected to by that primitive. In which way the environmental object is involved in the domain activity is given by the name of the relationship which has to be "filled in" by the model designer. This is similar to the way relationships are specified in Entity-Relationship Diagrams. However in contrast to relationships in ER-Diagrams an environmental-participation-relationship is graphically depicted as arrow. In order to be able to differentiate these only partially predefined link-primitives from the others in the final model their arrow consists of dotted lines. Furthermore the relationship name should be written in capital letters since the names of all predefined link-primitives are written in small letters. To avoid confusion it is pointed out that here we are concerned with domain activities in which the environmental object is involved but not with activities that are performed by the environmental object.

General Diagrammatic Symbol of the Environmental-Participation-Relationship Link-Primitive:

□ Connectivity:

There is no restriction for the application of the link-primitive *Environmental-Participation-Relationship*.

(6) The Node-Primitive "Environmental Activity"

□ **Definition:**

The primitive environmental activity is used to represent activities performed by environmental objects.

□ **Diagrammatic Symbol:**

□ **Applicable Link-Primitives:**

- *"activity_of":*

This predefined link-primitive is used to connect the environmental activity (tail of arrow) with an environmental object (head of arrow). The predefined semantics is that the activity is performed by the connected environmental object. Similar to the connectors *xxx_activity_of* for domain activities by this link-primitive an environmental activity can be connected to several different environmental objects.

- *"scheduled_in":*

By this predefined link-primitive temporal aspects with regards to performed activities can be considered. It can be used to connect the environmental activity (tail of arrow) with an primitive that is introduced later on and which is called time frame (head of arrow). The predefined semantics of this primitive is that the activity only is performed within the connected time frame but not outside that time frame.

□ **Connectivity:**

The primitive environmental activity can be connected to an unrestricted number of primitives of the type environmental object by the connector *activity_of*. Likewise it can also have an unrestricted number of connections to primitives of the type
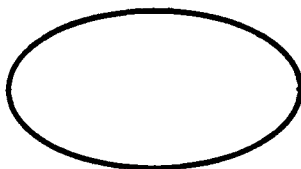
time frame by the connector *scheduled_in.*

(7) The Node-Primitive "Time Frame"

□ Definition:

The primitive time frame is used to represent certain periods of time which are of interest for a domain model for GenSIF. Time frames allow to consider temporal aspects. They define time intervals in which certain activities are performed.

□ Diagrammatic Symbol:

□ Applicable Link-Primitives:

- *"is_time_frame_in":*

This predefined link-primitive can be used to connect a time frame (tail of arrow) to another larger time frame (head of arrow). The predefined semantics of this connector is that the greater time frame (head of arrow) includes the smaller time frame (tail of arrow). Hence this link-primitive can be utilized to organize time frames in a "inclusion hierarchy".

□ Connectivity:

A primitive of type time frame may have an unrestricted number of connections to other primitives of type time frame by the application of the link-primitive *is_time_frame_in.*

6.4 Demonstration of Modeling Examples

In this section I continue with the introduction of the approach to a domain modeling notation for GenSIF. I illustrate in a few examples how the primitives allow us to model information that is relevant for a domain model. For that purpose the application domain "university department" has been chosen and the New Jersey Institute of Technology has served as the underlying "part of the real world" in the sense stated in section 5.4. It should be pointed out that the examples are based on a look at this "real world part" from the perspective of the rules of the Computer and Information Science Department. This has to be kept in mind to fully understand our choice concerning the predefined link-primitives in the presented examples.

As already stated, the application of the notation leads to a diagrammatic representation structure that could be viewed as a network. The components of the network are the modeling primitives that have been introduced in the previous section. It is the intention to use the presented modeling examples as "vehicles" to discuss in a less abstract manner the anatomy of these networks and their semantics.

The *italic* type style is used to refer to a modeling primitive. The `typewriter` type style is used to refer to an identifier in a given modeling example. The usual type style is used to discuss something in the real world.

In general the "backbone" of these diagrammatic networks is an object-based like representation of concrete and abstract objects of the application domain for which the primitive *object-class* can be applied. By the connector *category_of* specialization relationships between *object-classes* can be modeled. This means that the notation provides the abstraction mechanisms classification and specialization/generalization that help to make the description of the application domain manageable.

Classification in general allows us to group individuals into a class and to discuss properties of the class without referring to any members of the class (Borgida,

Greenspan, and Mylopoulos 1985).

Discussion of Modeling Example 1:

In the modeling example 1 (Figure 6.1) there are the *object-classes* PERSON, STUDENT, NEW STUDENT, REGISTERED STUDENT, and PROFESSOR which are roughly speaking some of the "containers" in which the concrete and abstract domain objects are "collected". The *object-class* STUDENT is associated with the *object-class* PERSON by the *category_of* relationship to represent that the objects contained in the former are a special category (a subset) of the objects contained in the latter more general one. The interpretation of the other *category_of* relationships between the *object-classes* is straightforward. The objects collected in the *object-class* NEW STUDENT is one subset and the objects collected in the *object-class* REGISTERED STUDENT is another subset of the *object-class* STUDENT. Both subsets are disjoint, i.e. they have no object in common. Furthermore the *category_of* connector between the *object-classes* PROFESSOR and PERSON represents that the objects of the former are a specialized subset of the latter.

Using the modeling primitive *role*, different views on the objects "contained" in an *object-class* can be modeled. Since there are almost always more than just one relevant point of view for certain objects, *object-classes* may be associated with several *role* primitives (See also the definition for the primitive *role*). Regarding modeling example 1 for instance, the objects contained in the *object-class* PROFESSOR are related to the three different *roles:* course instructor, thesis advisor, and researcher. The model designer can choose from a set of six different connectors the fitting one in order to establish the association. It can be said that such a connector declares a specific kind of role-ownership with regards to the connected *object-class.* The connectors are differentiated by the level of commitment towards the role (optional or mandatory) and the frequency of the role (one-time, repetitive, or permanent) (See also the definition of the primitive *role*
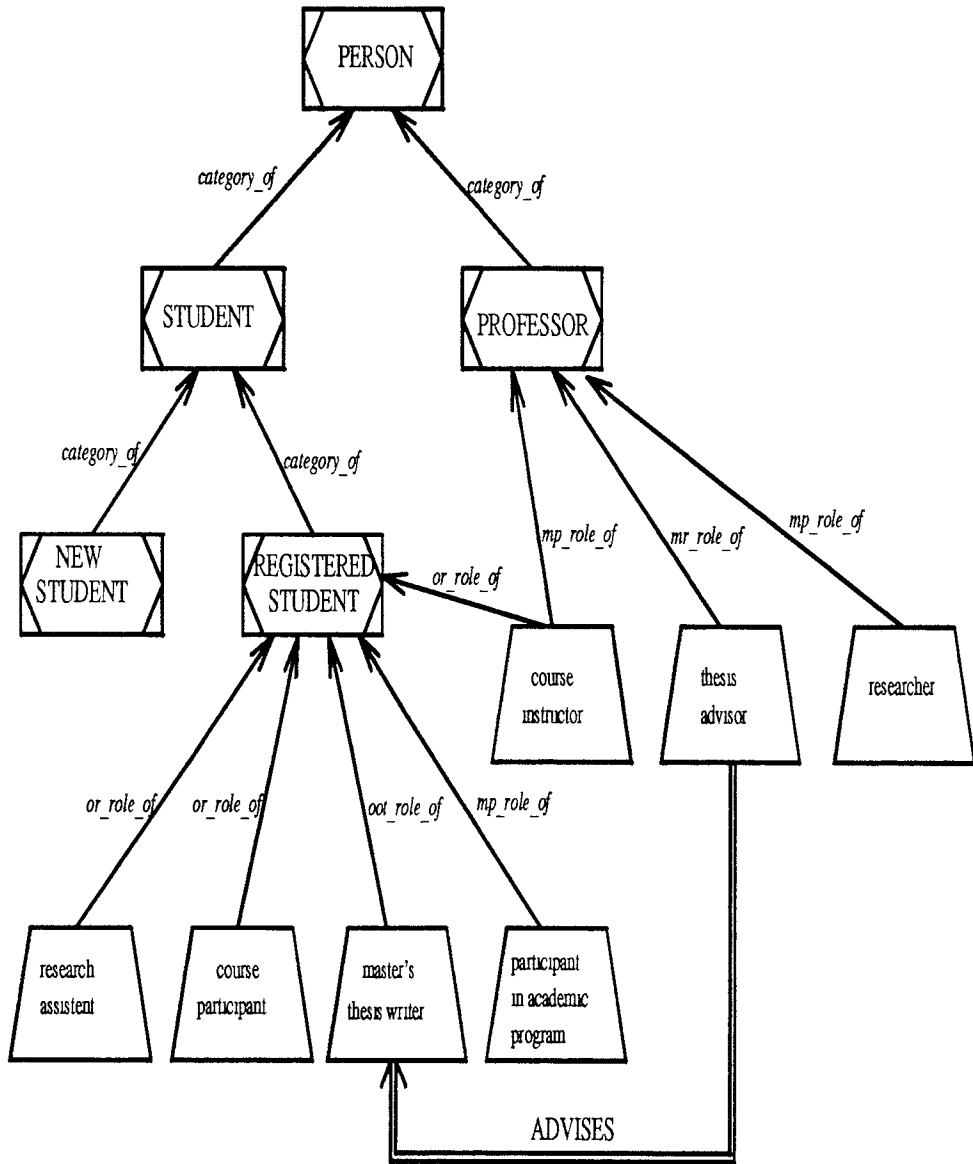
Figure 6.1: Modeling example 1.

). This in some sense is specification of meta information, i.e. information about information. To give an example, the *role* thesis advisor in modeling example 1 is linked by the connector *mr_role_of* (long form: *mandatory_repetitive_role_of*) to the *object-class* PROFESSOR. This reflects the fact (if we look at the department form the point of view of the rules of the department) that professors have the duty to be thesis advisor for more than just one time. In contrast to that, the connector *oot_role_of* (long form: *optional_one_time_role_of*) that has been chosen to link the *role* master's thesis writer to the *object-class* REGISTERED STUDENT represents that this role is not a duty for registered students and that a registered student can only be for one time a "Master's Thesis writer".

Since course instructor is not an exclusive role of professors with regards to the university department domain, the corresponding *role* in the example has also been linked to the *object-class* REGISTERED STUDENT by the connector *or_role_of* (long form: *optional_repetitive_role_of*). This shows another important feature of the notation. Since roles, which in our context correspond to a mission, job, duty, assignment or purpose in the real world, also can have different kinds of role owners, it is possible to link a *role* to many different *object-classes*. This contributes to a more realistic model and to better understanding of semantic contexts between domain objects and their roles. For instance the way we find the *role* course instructor connected to the *object-classes* REGISTERED STUDENT and PROFESSOR makes the gist of the semantic context between the objects in these *object-classes* and the *role* course instructor explicitly clear. Regarding that example, the semantic context is the following: While for professors being a course instructor is obligatory, for registered students this is optional.

The general advantage of such an explicit role primitive is that it reduces the redundancy of considering each role as a separate object and permits therefore more real world oriented modeling (See also the definition for the primitive *role*). This idea first has been invented in the Object Role Data Model (Bachman 1977).
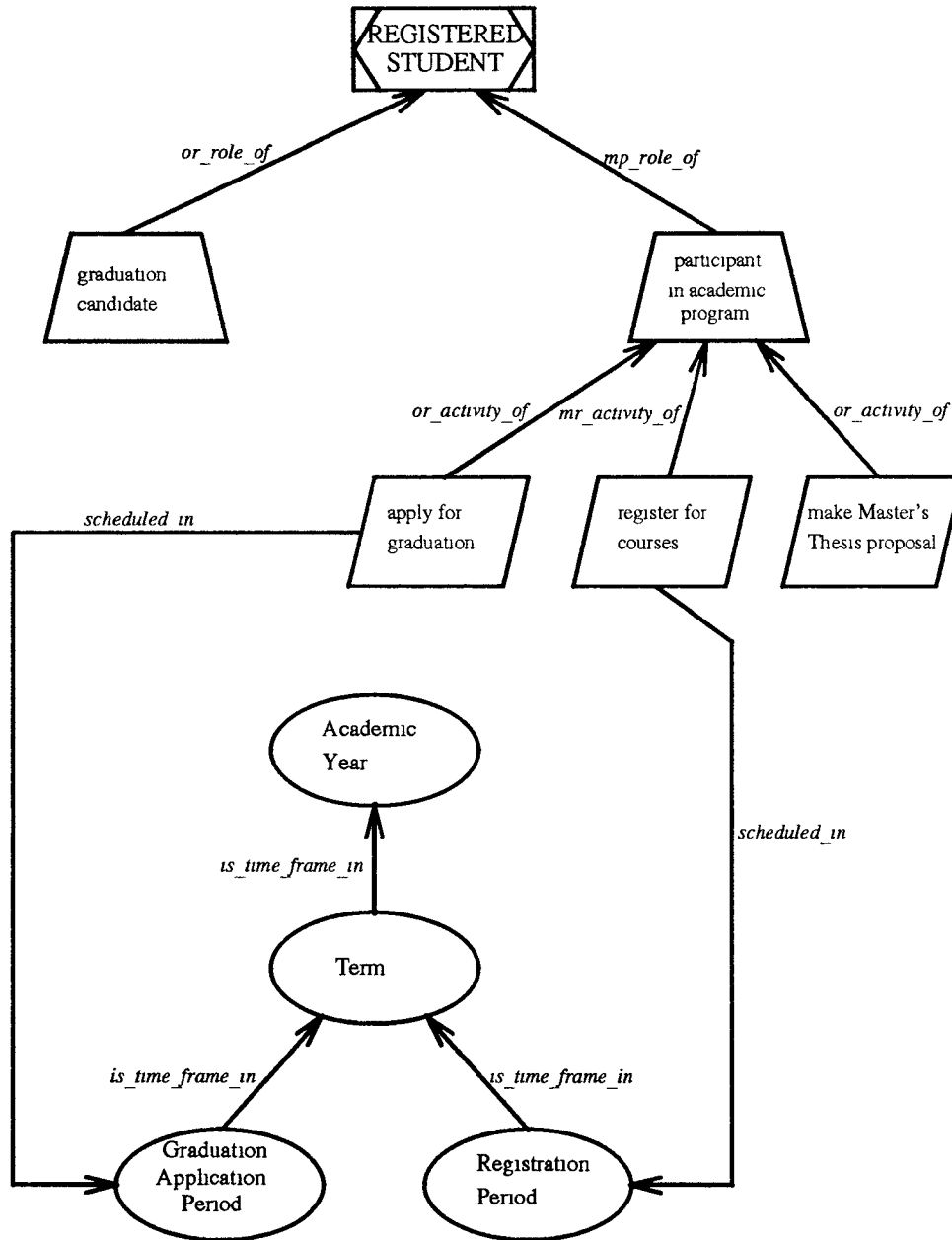
Figure 6.2: Modeling example 2.

The partially predefined *role-relationship* link-primitive (differentiated from other link primitives by a double line arrow and a relationship-name written in capital letters) has been used to represent that there is a relationship, called ADVISES, between the role thesis advisor and the role master's thesis writer. The interpretation which is straight forward is the following: The owner-objects of the *role* thesis advisor ADVISE the owner objects of the *role* master's thesis writer. By taking the connections into account these *roles* have to *object-classes* we get the following less abstract interpretation: Professors in their role as thesis advisor advise students in their role as master's thesis writer.

Discussion of Modeling Example 2:

Capturing of the behavior of objects, which are modeled by *object-classes* is realized by the primitive *activity*. In most cases *activity* primitives are associated with *role* primitives to structure the overall behavior of the objects in different sets of *activities*. In this case *activities* are accumulated by *roles* where the accumulated set of *activities* specifies the operational side of the *role*. However it also is allowed to associate an *activity* directly with an *object-class*.

In modeling example 2 (Figure 6.2) for instance there are three different activities represented which are performed by registered students in their particular role as participants in an academic program namely: *apply for graduation, register for courses, make Master's Thesis proposal*. Analog to the role connectors the notation provides a set of six different connectors from which the model designer has to choose the fitting one for the association between the *activity* and the corresponding *role* or *object-class* respectively. These activity connectors specify additional semantics regarding the level of commitment towards the activity (optional or mandatory) and the frequency of the activity (one-time, repetitive, or permanent).

Let us pick the activity apply for graduation for a closer look. The ac-

tivity is not mandatory since the completion of an academic program is completely up to the student. On the other hand, the activity might also be repetitive since if due dates are not obeyed, it can happen that the application for graduation has to be repeated. Therefore the connector *or_activity_of* (long form: *optional_repetitive_activity_of*) has been chosen to associate this *activity* with the relevant *role* (participant in academic program) of the *object-class* REGISTERED STUDENT. In contrast to that the *activity* register for courses is associated by the connector *mr_activity_of* (long form: *mandatory_repetitive_activity_of*) with the same *role* of the *object-class* REGISTERED STUDENT, since course registration must be repeated in each term by each participant of an academic program (if we look at the department from the point of view of the rules of the department). The third *activity* that is included in the modeling example 3 has the identifier make Master's Thesis proposal. The connector *or_activity_of* (long form: *optional_repetitive_activity_of*) has been chosen since making a proposal for a Master's Thesis is optional and eventually must be repeated if the proposal is not accepted.

This kind of modeling behavior in some sense reminds on the entity life-cycle diagrams of the Jackson's System Development Method (JSD) (Jackson 1983). In JSD life-cycle diagrams the sequence of activities associated with one entity corresponds to the sequence in which they are performed in the real world. This is not so in the representation structures which result from the application of the introduced notation. However, although only on a rough level, by the application of the modeling primitive *time frame* temporal aspects can be considered. If we look at modeling example 2 there is the *activity* apply for graduation connected to the *time frame* Graduation Application Period by the utilization of the link-primitive *scheduled_in*. This represents that the application for graduation has to be completed within a specific period of time. In a broader sense it can be regarded as a temporal constraint for the *activity* from which the connection to the *time frame* is realized. Since registration for courses also has to be completed within
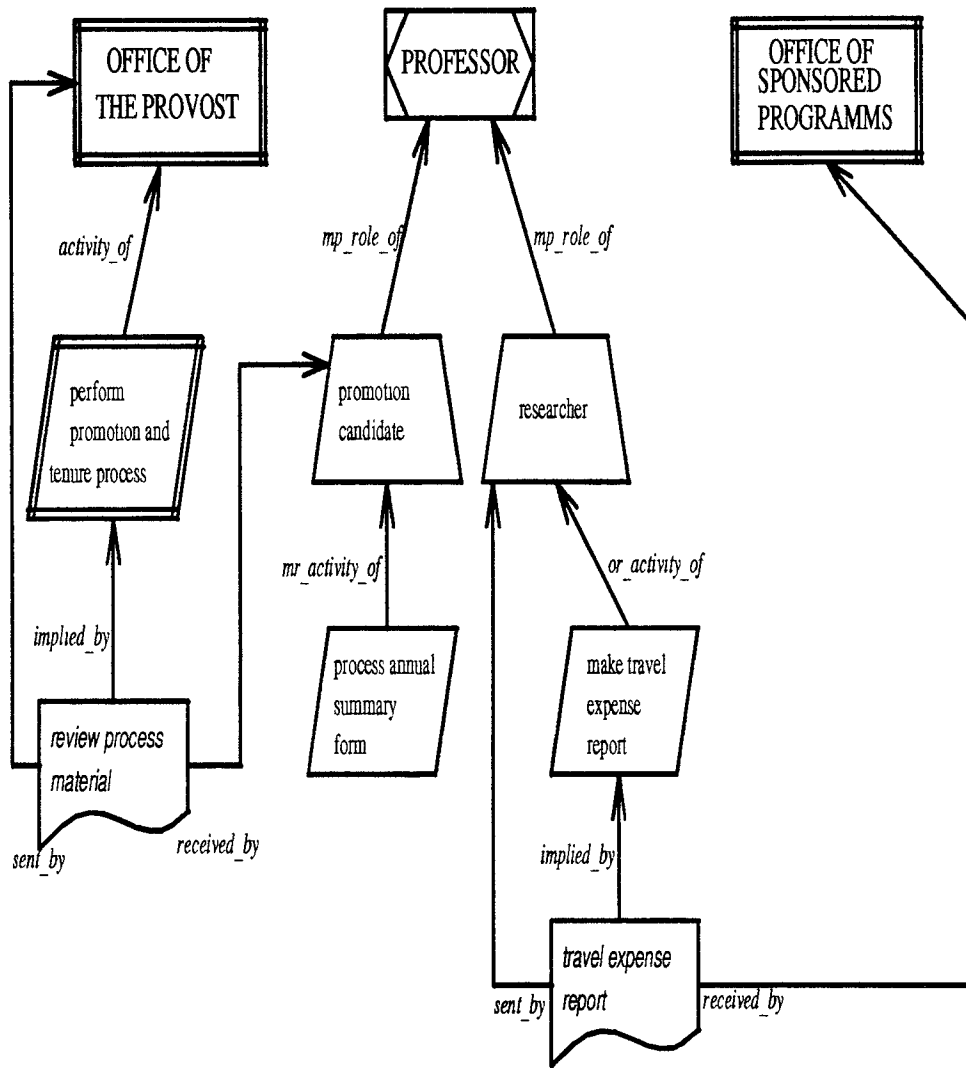
Figure 6.3: Modeling example 3.

a certain period of time the *activity* register for courses is associated with a *time frame* that has the identifier Registration Period.

Modeling example 2 also shows that *time frames* can be organized in an inclusion hierarchy. The *time frame* at the highest level of the hierarchy includes all other *time frames* on the lower levels. This hierarchical organization is accomplished by the utilization of the link-primitive *is_time_frame_in*. In the example the largest *time frame* at the top of the hierarchy is the *time frame* Academic Year. Since an academic year is divided in terms in the hierarchy of the example we find the *time frame* term on the second hierarchical level. Beyond that we have the *time frames* Graduation Application Period and Registration Period, each is connected to the *time frame* term by the predefined link-primitive *is_time_frame_in*. The interpretation of this is straight forward: An academic year includes academic terms and in each academic term there is a time period for the application for graduation and a time period for course registration as well.

The basic idea for this is that most application domains have their own particular time dimension, i.e. their application domain specific time frames that can include other smaller time frames. This approach allows to consider temporal aspects regarding habitual behavior of the (active) domain objects that perform activities. Two further examples for such top-level time frames are "business year" for the company domain and "production week" for the factory domain. Such time frames in some sense are generic since they are not only true for one specific period of time, e.g. academic year 1991/92.

Discussion of Modeling Example 3:

Let us take modeling example 3 (Figure 6.3) for a closer look how communication can be represented by the primitive *communication element* together with its applicable link-primitives. Here two *activities* are included that imply the exchange of certain information which is modeled by the utilization of these prim-

REGISTERED
STUDENT

PROFESSOR

OFFICE OF
SPONSORED
PROGRAMS

mp_role_of

mp_role_of

participant
in academic
program

researcher

mr_activity_of

or_activity_of

GUIDES

register for
courses

make
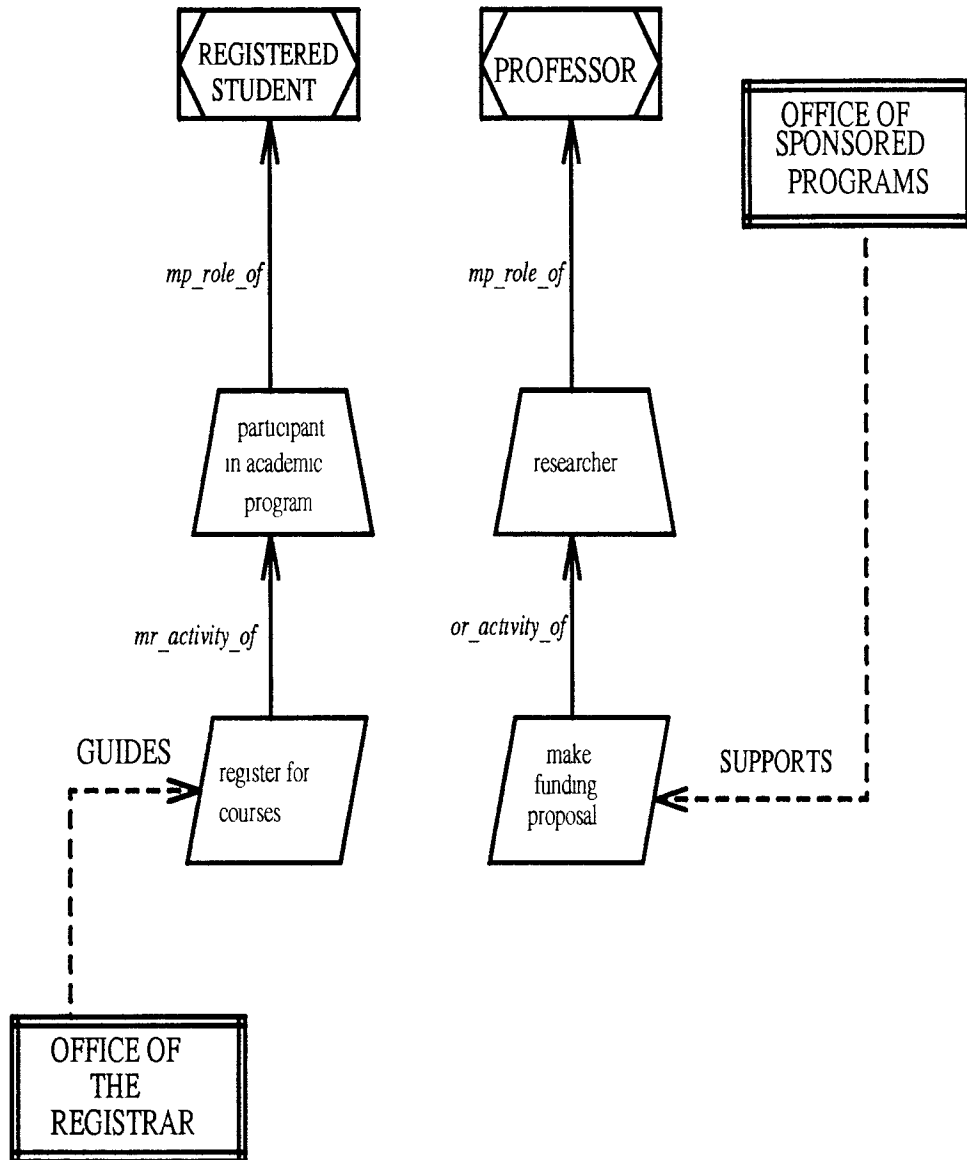funding
proposal

SUPPORTS

OFFICE OF
THE
REGISTRAR

Figure 6.4: Modeling example 4.

itives. The first *communication element* we want to discuss has the identifier `review process material`. The *implied_by* association to the *environmental activity* `perform promotion and tenure process` represents that this activity involves the exchange of review process material. The *sent_by* association to the *environmental object* `OFFICE OF THE PROVOST` specifies the sender of the review process material. The *received_by* association to the *role* `promotion candidate` defines professors in their role as promotion candidate as the receiver of review process material. The second *communication element* has the identifier `travel expense report`. The exchange of it is implied by the *activity* `make travel expense report`. The objects in the *object-class* `PROFESSOR` which are the owners of the *role* `researcher` are defined as the sender. The *environmental object* `OFFICE OF SPONSORED PROGRAMS` is modeled as the receiver of the *communication element* `travel expense report`.

The modeling example 3 also shows the utilization of the modeling primitives *environmental object* and *environmental activity*. Since the "Office of the Provost" is not considered being a part of the application domain "university department", it has been modeled by the usage of the primitive *environmental object*. In section 5.1 principle III the consideration of such environmental objects is discussed in more details. Activities that are performed by those environmental objects if they need to be considered are represented by the primitive *environmental activity*. Hence `perform promotion and tenure process` is represented by the primitive *environmental activity*.

Discussion of Modeling Example 4:

Modeling example 4 (Figure 6.4) shows the application of the partially predefined *environmental-participation-relationship* link-primitive. The link-primitive is partially predefined since in contrast to completely predefined link-primitives the name of the relationship has to be given by the model designer. The relationship-name for
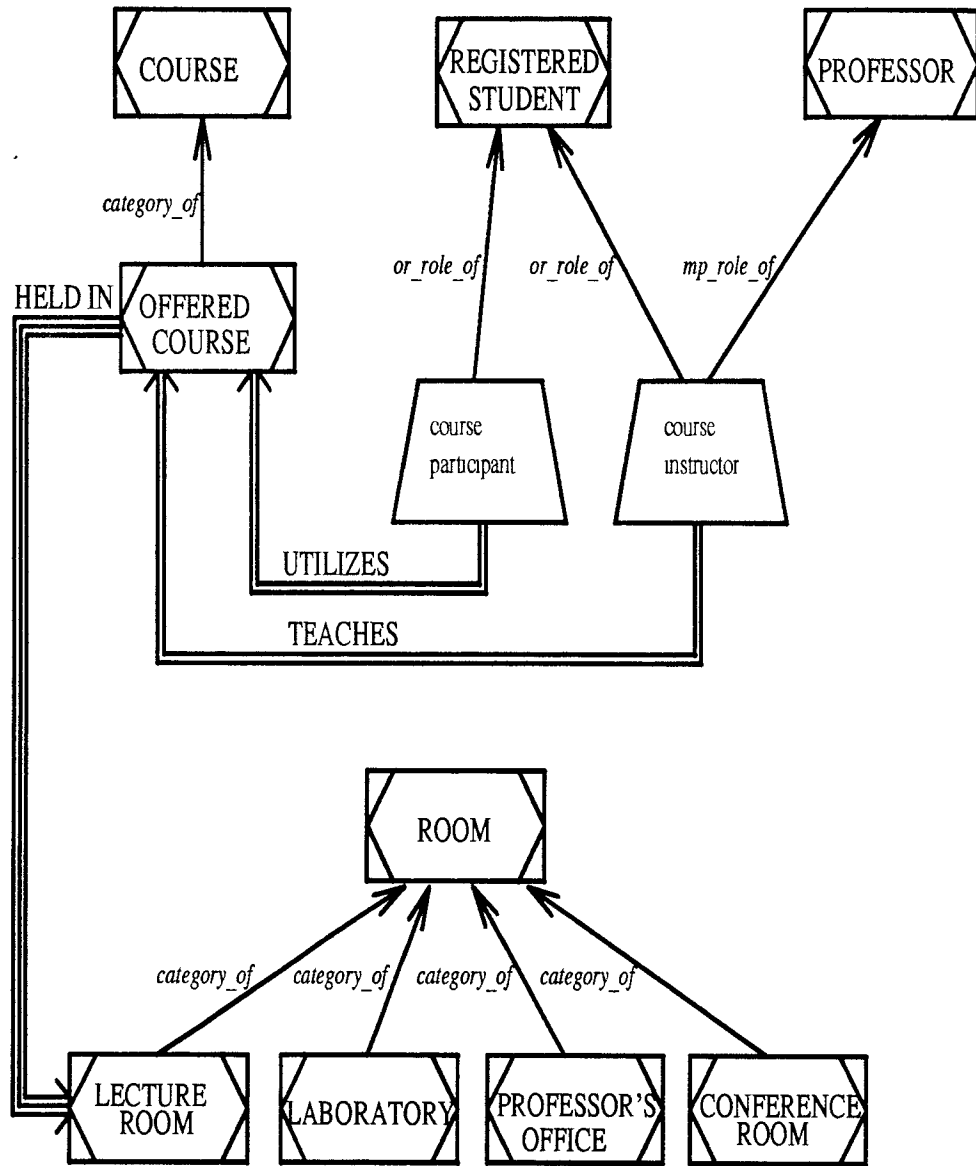
Figure 6.5: Modeling example 5.

the association between the *environmental object* OFFICE OF SPONSORED PROGRAMS and the *activity* make funding proposal is SUPPORTS (To avoid confusion it is pointed out that here we are concerned with a domain *activity* and not an *environmental activity*). This represents that the "Office of Sponsored Programs" supports the elaboration of funding proposals which is performed by professors in their role as researchers. The relationship-name for the association between the *environmental object* OFFICE OF THE REGISTRAR and the *activity* register for courses is GUIDES. This represents that the "Office of the Registrar" is responsible for the administration of the course registration.

Discussion of Modeling Example 5:

In modeling example 5 (Figure 6.5) the utilization of the partially predefined link-primitive *object-class-relationship* is introduced. This is done within the context of the consideration of some abstract *object-classes* which we can identify in the "university department" domain. First of all we have the abstract *object-class* OFFERED COURSE which is defined as a subset of the *object-class* COURSE by the *category_of* connector. Furthermore we have the *object-classes* LECTURE ROOM, LABORATORY, PROFESSOR'S OFFICE, and CONFERENCE ROOM, which are all disjoint subsets of the *object-class* ROOM. Between the *object-class* OFFERED COURSE and the *object-class* LECTURE ROOM there is an *object-class-relationship* denoted with the name HELD IN. The interpretation of this part of the given example is straight forward: offered courses are held in lecture rooms. In addition we find the *object-class* REGISTERED STUDENT and the *role* course participant that is connected to it. Furthermore there is the *object-class* PROFESSOR and the *role* course instructor that has been linked to the latter and the *object-class* REGISTERED STUDENT as well. Since that slice of the modeling example also appears in some of the previous examples, there is nothing new so far. However the utilization of the predefined *role-relationship* link-primitive (double line arrow) to connect a *role* with an *object-class* has not

been presented in the previous examples. In modeling example 5 this is demonstrated twice. In its first application to connect the *role* `course participant` to the *object-class* `OFFERED COURSE` it has been used to capture the information that course participants utilize offered courses. In the second application where it connects the *role* `course instructor` to the *object-class* `OFFERED COURSE` it represents that course instructors teach offered courses.

6.5 Contributions of This Experiment to the GenSIF Project

As mentioned before, what has been presented in this chapter should be understood as a first approach to a notation that fits the specific needs of domain modeling within the GenSIF framework. We in the research group for systems integration at the New Jersey Institute of Technology (spring term 1992) have used this approach to drive our reasoning about domain analysis as component of GenSIF. It should be pointed out that we had a controversial discussion with an open end, concerning the appropriateness of the notation introduced in this chapter. A critical discussion of the notation is included in subsection 8.2.2.

Nevertheless, the notation and the presented modeling examples respectively, advanced our discussion of domain modeling for GenSIF. Hence it can be claimed that one contribution of this experiment is the preparation of an adequate "vehicle" for a further and more concrete discussion of domain modeling within GenSIF.

Based on the experience of the utilization of the notation in our research group we also believe that beginners who work in that problem area might find a quicker mental access to the problem through this documented experiment. Therefore in a broader sense the notation might be called a "catalyst" for the familiarization with domain modeling for GenSIF.

In addition to these contributions, in the following paragraph the author strives to

discuss the question if there is some more potential in the introduced notation.

Is There Some More Potential in the Result
of This Design Experiment?

As the reader might have already guessed, it is the personal opinion of the author that the introduced approach to a domain modeling notation under certain conditions has some more potential then only the contributions listed before. In addition it might be an adequate "platform" for further research in that area. In the following the main aspects within that context are discussed.

Appropriate Notation for a Step Preceding Computer Based Domain Modeling:

An often referenced domain modeling language in the relevant research literature is the requirements modeling language RML developed by Sol Greenspan as part of the Taxis project at the University of Toronto (Greenspan 1984). An RML requirements model is meant to serve as a bridge between a requirements definition expressed in SADT (Ross 1977) and a system design expressed in Taxis. Greenspan views SADT as an appropriate notation for capturing intuition about the system, leading to a formal representation in RML. Although in domain modeling for Gen-SIF we are not concerned with any system design at all, it is at least a quite reasonable idea to precede the computer based modeling task by a manual step. For this manual modeling step the introduced notation might be an appropriate notation. As shown in the provided modeling examples it can be used to manually capture the relevant domain information on the required conceptual level.

"Criteria" for the Selection of a Domain Analysis Support Tool for GenSIF:

In a broader sense the introduced modeling notation can be used as a "checklist"

for the selection of a tool that is appropriate to domain analysis as component of GenSIF. Certainly it would be a mistake to strictly require that such a tool has to facilitate a modeling formalism that exactly corresponds to our notation. Instead the author would like to suggest that the modeling formalism should be understood as a "pointer" in the correct direction. This basically means that every candidate tool that allows to capture the same information as the introduced notation can be regarded as a "hot candidate tool". The issue concerning an appropriate tool for the entire domain analysis concern within GenSIF is discussed in the next chapter.

"Heart" of a Research Tool Implementation of a Domain Analysis Support Tool for GenSIF:

Currently there are too many open questions regarding the introduced notation. But eventually after its completion and improvement it might be utilized as the "heart" of a research tool implementation of a specific domain analysis support tool for GenSIF. As it can been inferred from the discussion given in section 6.2, implementation issues have been considered in the general approach to the notation.

CHAPTER 7
SOME REMARKS ON A DOMAIN ANALYSIS
SUPPORT TOOL FOR GenSIF

The purpose of this chapter is to discuss, the essential aspects regarding a computer based support tool for domain analysis as concept of GenSIF. The objective is to provide questions and interdependencies to consider if one wants to select or built a domain analysis support tool for GenSIF.

In the first section, a broad definition for such a tool is provided. Here also the three fundamental aspects regarding such a support tool that are investigated in the succeeding sections are identified.

7.1 A Framework for the Required Type of Tool

Although there are still some open questions regarding domain analysis as concept of GenSIF, it is possible to identify the essential aspects of a domain analysis support tool for GenSIF. Within this thesis such tool is regarded as a tool that facilitates the following:

- domain model design

- domain model manipulation (evolution and refinement)

- domain model utilization

Although the last mentioned point, i.e. the utilization of the domain model, actually does not belong to the domain analysis process any more, it would be a mistake to exclude this issue from this discussion. The reason is that the specific way the domain model is utilized in GenSIF also has some implications for the support tool, as explained in section 7.3.
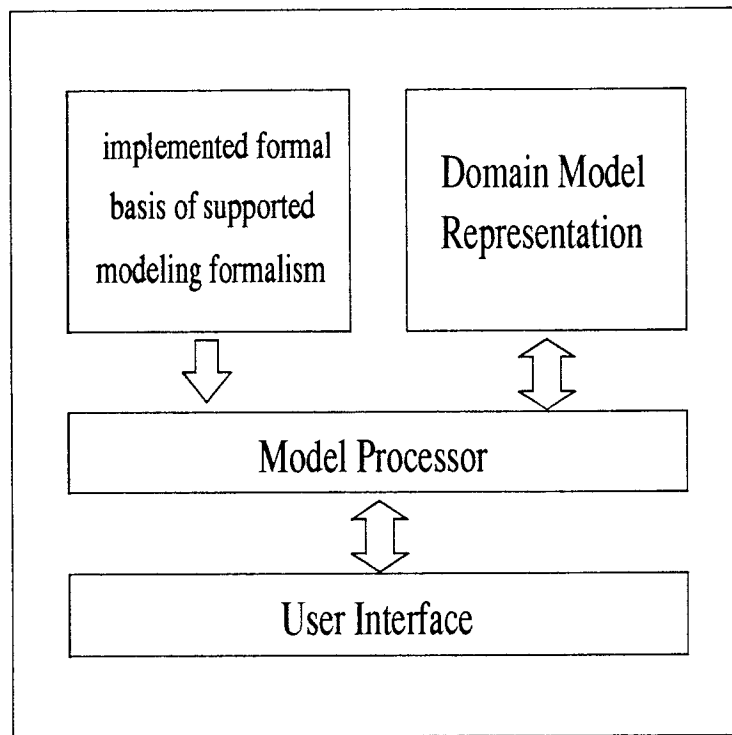
Figure 7.1: Conceptual schema of a domain analysis support tool.

In figure 7.1, in which the basic structure of such a tool is depicted graphically, I have called the component that handles the processing of the model (design, changes, queries, ... etc.) "Model Processor". This "Model Processor" can simply be imaginated as a set of functions that can be applied to design and manipulate the domain model according to the implemented formal basis of the underlying modeling formalism. Furthermore it includes "services" for the utilization of the domain model itself.

Within this framework the component that has been called "implemented formal basis of supported modeling formalism" is that facility which enables the "Model Processor" to generate, process, and to interpretate the internal representation of the domain model.

Hence the component "Domain Model Representation" is the computationally processable internal representation of the domain model.

The user interface in this framework is the facility which allows to interact with

the model processor and which is responsible for the presentation of the domain model to the user of the tool.

It should be pointed out that it is not claimed that this is the only way to describe a domain analysis support tool. Actually it is what the author has used for his reasoning about the requirements for a tool that supports domain analysis for GenSIF. On this basis, three fundamental aspects that need to be discussed in more detail have been identified:

(1) Modeling Formalism

The first aspect, certainly the most important one, is concerned with the requirements for the modeling formalism whose formal basis should be implemented in such a tool. This is the topic of section 7.2.

(2) Computational Processability

The second aspect is that computational processability of the domain model determines the power of what has been called the "Model Processor". The computational processability itself depends on the implementation of the modeling formalism. A processable syntax of the modeling formalism permits machine manipulability of the domain model up to a certain degree. But only a processable semantic permits powerful operations such as semantic retrieval, or consistency checking. Especially within the context of the domain model utilization a processable semantic seems to be required, as discussed in section 7.3.

(3) Presentation of the Domain Model

The last aspect that is considered here concentrates on the user interface, the presentation of the domain model to the user. Since a domain model within the GenSIF framework is massively used by the users in their different tasks, the presentation of the model has also to be considered as an important criteria. Section 7.4 provides a discussion of this criteria.

7.2 On the Modeling Formalism That Should be Facilitated

In general a modeling formalism "... *describes an epistemology that accounts for how a concept has meaning within the model as well as what it denotes in the world* (Carasik, Johnson, Patterson, and von Glahn 1990)".

It basically can be compared to a data model "... *that provides a formal (notational and semantic) basis for tools and techniques used to support data modeling* (Brodie 1982)," but with the difference that it has to support domain modeling.

The central goal of such a modeling formalism is to provide facilities for gathering and representing the selected domain phenomena in a natural and convenient fashion, and at the same time to organize and structure the representation so that it can be easily accessed and (re-)used. This implies two questions to be answered regarding the modeling formalism in a tool for domain analysis for GenSIF:

1. Which level of expressive power is necessary and sufficient?
2. Which facilities for structuring and organizing the domain model are necessary and sufficient?

(1) Expressive Power

What is called here expressive power basically regards to the question what should be describable with the modeling formalism. The modeling notation introduced in the previous chapter is one possible answer to that question. Everything expressable with that notation should also be expressable with the modeling facilities supplied by the tool. That means, for example, that a tool with an underlying modeling formalism that is appropriate for the static aspect of the application domain, but lacks expressive power for the description of the two other relevant aspects of the domain is not sufficient.

(2) Facilities for Organizing and Structuring the Domain Model

Domain models in GenSIF should be organized in a way that allows to identify and to understand the general structure of the environment of the integrated system (Rossak 1992a). For that reason the modeling formalism has to provide structuring facilities that allow to organize the domain knowledge in such a way, to use an old German saying, that enables the user of the model "to see the forest despite the many trees". Expressed in a more academic way, these structuring facilities are important with respect to the intellectual manageability of the large amount of highly interdependent information about different aspects of an application domain that have to be integrated into a domain model.

For a long time it has been asserted that abstraction is the best tool we have toward the intellectual manageability of complex descriptions. An abstraction mechanism is a conceptual or linguistic mechanism that allows certain information to be highlighted while suppressing other information (Mylopoulos, Borgida, Greenspan, and Wong 1984). Another, basically same definition but with the data handling side as point of view is given in (Garg 1988), where abstraction mechanism is defined as the means by which information can be stored and retrieved from an information structure at different levels of detail and from different perspectives.

In software engineering, abstraction is usually equated with the suppression of design decisions or implementation details. Since in this thesis a domain model has been already characterized in section 5.2 as similar to a conceptual model of the application domain, it is appropriate to have a look at the research area of conceptual modeling. Abstraction mechanisms have been intensively studied there, for example in (Brodie, Mylopoulos, and Schmidt 1986; Bubenko 1983; Olle, Sol, and Verrijn-Stuart 1982). To introduce the 14 different types, or classes, of structuring principles provided in (Kangassalo 1983) certainly goes beyond the scope of this thesis. However from the approach to a specific modeling notation for GenSIF introduced in the preceding chapter it can be inferred that classification,
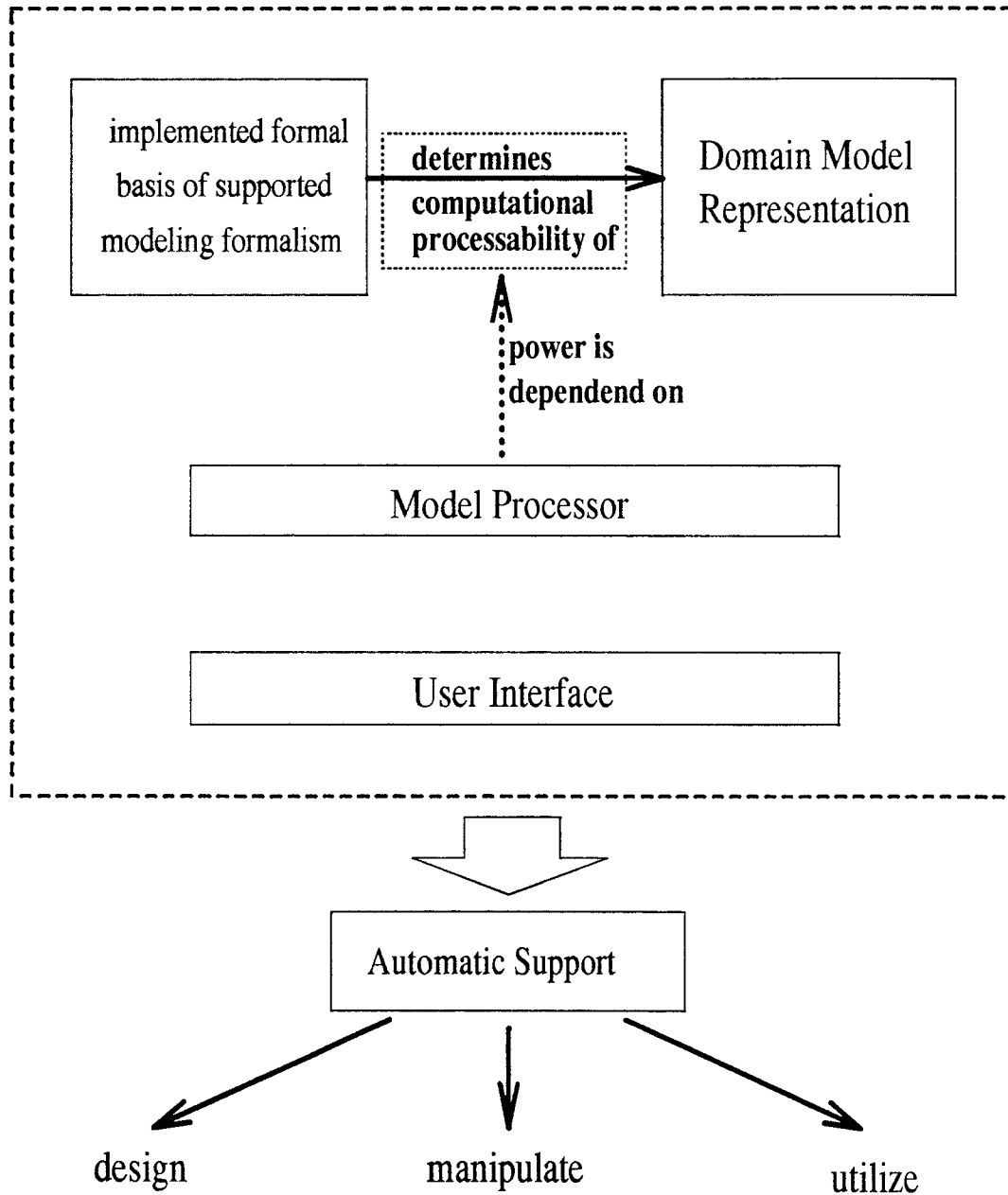
Figure 7.2: Relationship between modeling formalism implementation, computational processability and automatic support.

generalization and aggregation are necessary structuring facilities that should be provided by the formalism.

7.3 Computational Processability and Automatic Support

The level of support to design, manipulate, and utilize the domain model provided by the support tool depends on the power of what has been called the "Model Processor" in section 7.1. The power of the "Model Processor" itself depends on the computational processability of the domain model (Figure 7.2). What is addressed in this discussion is the question how much of the information captured in the domain model can be interpreted by the computer. The answer to that question is determined by the computational tractability of the modeling formalism; the modeling formalism implementation respectively. If the implementation of the formalism just covers its syntax, then only the syntax of the domain model is processable. However if in addition also the semantics of the formalism is implemented, then also the semantics of the domain model is processable and interpretable by the computer (up to a certain degree). The difference between both can be explained by looking at the types of queries that could theoretically be facilitated by the tool. A consequence of a purely syntactic implementation, where the terms used have no description (like in relational databases or keyword systems), is that the users of the tool are forced to provide the exact identifies that are used in the domain model. By means of an implementation of the semantics of the modeling formalism, the user can describe a query in terms which may be different from the exact terms under which the desired domain information is stored, as long as the meaning is similar.

In the following classification I have attempted to clarify the relationship between the level of support for designing, manipulating and utilizing the domain

model within the context of GenSIF and the corresponding requirements for the processability of the domain model.

Although an interesting point, the impact on the workload and qualification of the involved system engineers and system developers is not included.
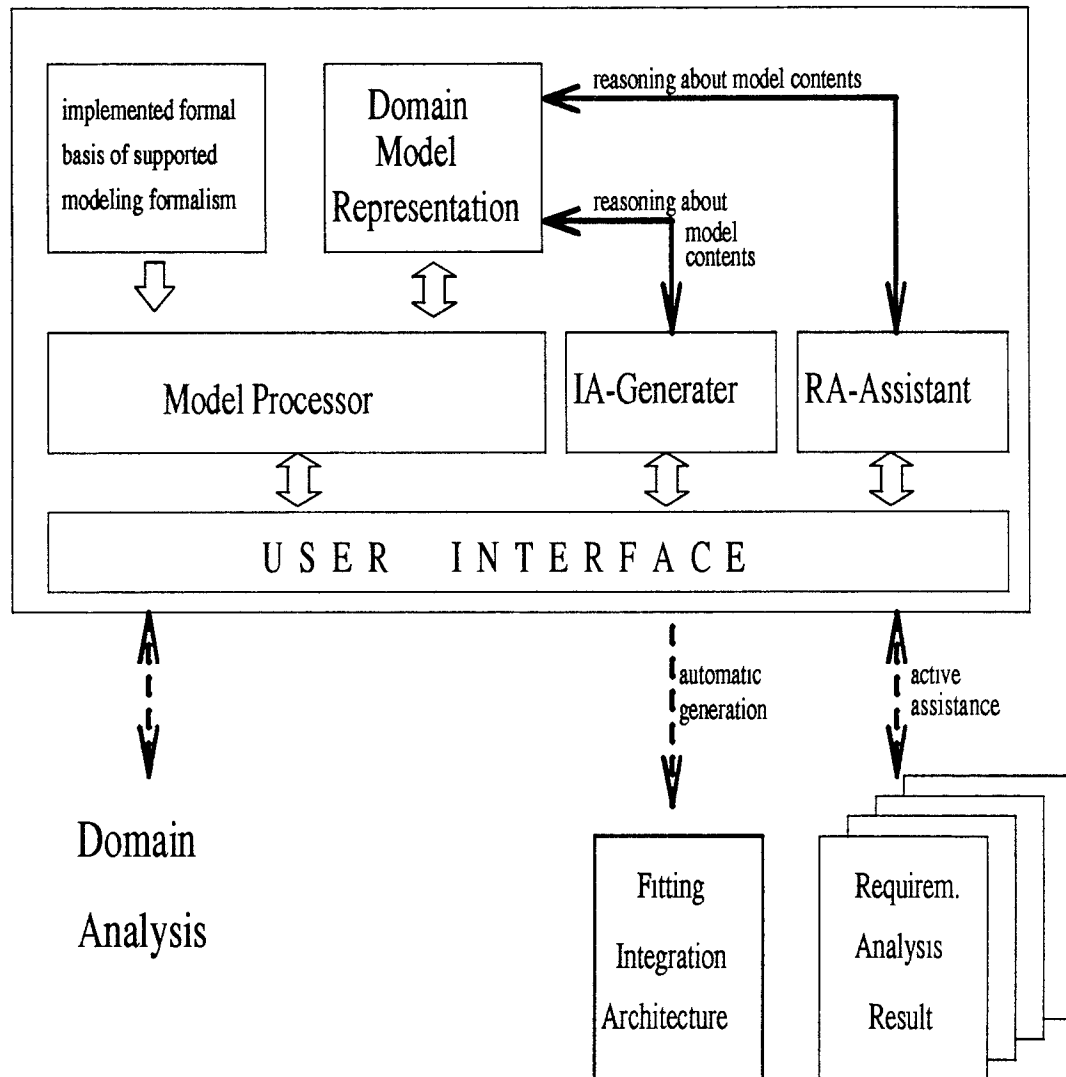
"Trivial" Automatic Support:

The level of automatic support that is regarded here as "trivial" support incorporates those functions that are performed based on simple pattern matching mechanisms. The processability of the domain model simply covers the syntax but not any semantics. The semantics of the domain model contents is not interpretated in any function at all.

This outline of automatic support perfectly matches to that typically provided by Hypertext systems, since the goal of Hypertext users is often to capture an interwoven collection of ideas without regard to their machine interpretability (Conklin 1987).

Some examples for concrete functions which belong to that level are: searching, browsing, focusing, "trivial" query processing, ...etc.

"Intelligent" Automatic Support:

In contrast to the previous level of automatic support I have called this level "Intelligent" Automatic Support since the processability of the domain model is enriched by the semantics of its content up to a certain degree. This basically means that the content of the domain model partially can be interpretated by the computer. Hence automatic support is enlarged by "intelligent" functions, such as typically provided by knowledge representation systems. Some examples for concrete functions which belong to this level are: semantic retrieval, consistency checking, "explanatory" functions, "active modeling assistance", ...etc.

implemented formal basis of supported modeling formalism	Domain Model Representation	reasoning about model contents

reasoning about model contents

reasoning about model contents

Model Processor

IA-Generater

RA-Assistant

USER INTERFACE

Domain

Analysis

automatic generation

active assistance

Fitting Integration Architecture

Requirem. Analysis Result

IA-Generater: Integration Architecture Generater

RA-Assistant: Requirements Analysis Assistant

Figure 5.5: Computer aided systems engineering and systems development with GenSIF

Scenario for the Future:
"Computer Aided Systems Engineering and Systems Development with GenSIF"

What is presented here is "a look beyond the rim of the plate". If computer programs are used to land airplanes, to control nuclear power plants, or "just" used to control production plants, why shouldn't we think about the possibility to have a domain analysis support tool that is able to derive automatically the integration architecture and give active assistance to the requirements engineer as well. In fact, Fickas "KATE" system (Fickas 1987) and the "MIT Requirements Apprentice" system (Rich, Waters, and Reubenstein 1987) are research efforts which basically have similar objectives. (Similarities can also be seen between the former and what is attempted to realize in the research area of automatic programming (Barstow 1985).)

The two additional fundamental components such a tool for computer added systems engineering and system development with GenSIF would require are a data base of predefined integration architectures and a rule base, in which knowledge concerning the question which integration architecture fits to which application domain is encoded in rules (Figure 7.3).

In order to derive the fitting integration architecture, such a tool would require a high level of processability of the semantics of the domain model and support reasoning about the domain structure. Regarding processability of the domain model, such a tool would require a high level processability of the semantic of the domain model that must allow to reason about it in order to derive the fitting integration architecture.

7.4 Domain Model Presentation - The User Interface

A good modeling formalism and high processability of the domain model are important aspects for a domain analysis support tool as introduced in section 7.1. However if the user interface is bad in the sense that the domain model is presented in a "poor" manner most of the gained advantages would be lost.

The domain analysis support tool provides the required facilities to build and maintain a complex domain model and to utilize it for the purposes described in chapter 3. There should be no doubt that already for that goal a good model presentation is of importance. But an even stronger argument for a high level presentation can be seen with regards to the aspect that the domain model is communicated in a group of persons, probably even a changing group of persons, over a long period of time. Therefore, it is a necessary requirement for such a domain analysis support tool to present the domain model in a manner that facilitates understandability and semantic interpretability of the model.

A text-based only presentation of the model to the user certainly is not enough. What is required is visual and graphics support, which allows the domain modeling engineer to interact with and even directly manipulate graphical output. Significant visual support, such as browsing, tracing and navigating facilities, as provided by "state of the art" commercial knowledge representation systems and hypertext systems are important facilities regarding the user interface of the tool. Such facilities contribute to the intellectual manageability of the highly interdependent and complex information regarding the application domain.

Especially with regard to the utilization of the domain model by requirements engineers of application projects, facilities to "browse and navigate through the model" and to trace back particular model components play an important role. Since a requirements engineer might be interested only in a particular view on the modeled real world part, a facility to concentrate on one particular point of

view and to cut out the others is desired as well. Likewise a facility to focus on "slices" of the domain model is an important feature that would ease the task of the requirements engineer within the GenSIF framework. This would include facilities to focus on one particular aspect of the application domain (e.g. only the static structure of the application domain), and to "fade out" others.

In general all these facilities are concerned with the ability to instruct the domain analysis support tool to present a reduced domain model. With regard to this, the part of the model that should not appear in the reduced domain model should be flexible and easy describeable by the user.

CHAPTER 8
FINAL DISCUSSION

The contents of the thesis are outlined in the summary given in section 8.1, where the main areas I have addressed are described. In each of the areas, currently open research questions are identified and discussed in section 8.2. Finally, in section 8.3, I provide my conclusions and propose how the research this thesis is concerned with could be continued.

8.1 Summary

The research reported in this thesis is a first effort to elaborate how domain analysis should look like if the overall goal is systems integration. With regards to this area, it is focused on the special needs of the GenSIF framework for systems integration as proposed in (Rossak and Ng 1991; Rossak 1992b; Zemel 1992).

The two main areas addressed in the thesis are:

1. The elaboration of domain analysis as component of GenSIF, i.e. the description of important aspects of domain analysis within the GenSIF framework.

2. An approach to a domain modeling notation that fits the specific needs of the GenSIF framework.

Furthermore based on the gained insights some remarks regarding a domain analysis support tool for GenSIF are summarized.

8.2 Open Research Questions

8.2.1 Elaboration of Domain Analysis as Concept of GenSIF

In our current approach to domain analysis within GenSIF we motivate the design of a general domain model. Hence information regarding quantities and exact dates with respect to temporal aspects are not considered at all. Currently it is an open research question if such a general domain model can contribute as a "decision support tool" to the selection of the fitting integration architecture. (See also subsection 5.4.2)

8.2.2 Approach to a Specific Domain Modeling Notation for GenSIF

As it has been pointed out several times throughout this thesis the presented approach to a specific domain modeling notation for GenSIF has to be regarded as a first attempt to such a specific notation. Hence the notation introduced in chapter 6 has some shortcomings which are discussed in this section.

First of all, despite the possibility to integrate several different views on domain objects in one model by the utilization of the modeling primitive *role*, the notation does not really allow to integrate different views on the relevant real world part. The reason for this is that a *role* can only be connected to a relevant *object-class* by a single *xxx_role_of* link-primitive and the final choice for the link-primitive depends on the way the model designer looks at the relevant real world part. To make this clear with an example it is referred to modeling example 1 (page 60, Figure 6.1). Here for instance the connector *mp_role_of* (long form: *mandatory_permanent_role_of*) has been used to link the *role* thesis advisor to the *object-class* PROFESSOR. This is based on looking at the domain "university department" from a point of view of the rules of the university department. However if we would look at the domain

from the point of view of a professor as a human being we would rather choose the connector *or_role_of* (long from: *optional_repetitive_role_of*). Despite the duty to accept students as his/her Master's Students, certainly "thesis advisor" is not a role which is truly mandatory from the point of view of a professor.

The same is also true for the utilization of the modeling primitive *activity*. Here the number of connections an *activity* may have to a relevant *role* or *object-class* respectively also is restricted to exactly one connection but not more. Let us take modeling example 2 (page 62, Figure 6.2) to discuss this in more details. Here the *activity* `make Master's Thesis proposal` is linked to the *role* `participant in academic program` by the connector *or_activity_of* (long form: *optional_repetitive_activity_of*). This particular connector has been selected based on looking at the real world from the point of view of the rules of the department. Since there is not any rule included which states that participants of an academic program (here the Master's Program) have to make a Master's Thesis proposal this activity is represented as an optional activity in the example. But it has to be considered that if we would look at the real world part from the point of view of a student who is participating in the Master's Program we would rather choose the connector *mr_activity_of* (long form: *mandatory_repetitive_activity_of*). This can be claimed since for a student the preparation of a Master's Thesis proposal is an absolutely "must-activity", i.e. a mandatory activity if we assume that his/her intention is the completion of an academic program.

This shortcoming could be removed by allowing more than only one single link-primitive between the relevant pair of primitives where each connection corresponds to a specific view at the real world. However this would imply that each connection has to be identifyable regarding the underlying point of view. This means that these link-primitives must also "carry" information with respect to the point of view behind them. Another alternative is to simply acknowledge that the notation cannot handle different points of user views in one comprehensive representation

structure. Hence, different views have to be considered in different graphical models ("slices").

It is also an open issue if the link-primitives applicable to the node-primitives *role* and *activity* are appropriate and sufficient for the design of realistic domain models. As pointed out in section 6.3 they have been selected from a practical point of view. I am quite sure that they need to be modified qualitatively as well as quantitatively after we have gained more experience regarding our domain modeling concern. The problem within this context is to find the right balance regarding the trade-off between modeling guidance towards the required kind of model and limitation of freedom of expression for the model designer.

The understandability, i.e. readability, of the notation could be improved if the predefined link-primitives would be graphically differentiated as well. A differentiation criteria on a top level could be the issue whether a connection represents a logical (*xxx_role_of, xxx_activity_of, is_scheduled_in* , ... etc.) or physical (*sent_by, received_by*) connection. A criteria on a lower level could be the issue whether the *role* or *activity* respectively is mandatory or optional. This would make it redundant to have each link-primitive associated with its name.

A big shortcoming of the introduced notation is the lack of facilities to model complex temporal aspects. At the current status the notation only supports the consideration of time intervals in which certain activities are performed in the domain. This is similar to the specification of temporal constraints regarding the particular time interval within a certain activity has to be performed. These time intervals are mapped into the modeling primitive *time frame* which can be organized in an inclusion hierarchy. This makes it fairly easy to incorporate some temporal aspects of activities on a rough level. However, to provide only this facility certainly is not enough. Other connections between time frames which might be of interest are for example: BEFORE, EQUAL, SYMMETRIC OVERLAP, ASYMMETRIC OVER-LAP. Research work on this issue is provided for instance in (Balour, Anderson,

Dekeyser, Wong 1981; Allen 1981; Bubenko 1980).

Furthermore, the notation lacks a facility to consider temporal aspects that include instantaneous events which are connected to some reactions within the domain. What is required is something like a mechanism that allows to generate events that are the triggers for some reactions. However, the incorporation of such a mechanism without "breaking-down" the conceptual level of our modeling approach we have so far seems to be non-trivial.

The notion of inheritance has not been discussed at all throughout the introduction of our approach to avoid confusion. However, the issue if we need inheritance has to be investigated. Regarding our approach to a modeling notation this would not imply a modification of the primitives. However, the incorporation of inheritance in our approach would change the way our networks have to be interpretated.

8.2.3 Selection of a Domain Analysis Support Tool for GenSIF

There are several types of tools which are based on different technological approaches that more or less match to the description of a domain analysis support tool as given in chapter 7. These are the following one:

- Existing Tools for Domain Modeling

- Knowledge Representation Systems

- Hypertext Systems

- Special Modeling Languages for Software Engineering

- Object Oriented Analysis and Design Systems

The question now is which is the best choice with regards to its application as a domain analysis support tool for GenSIF.

There are only a few specific domain modeling tools on the market. Most of these tools are oriented towards software-reusability. Hence these tools facilitate a conceptual as well as constructive kind of analysis. The set of primitives which is provided by these tools mostly focuses on the static structure of an application domain. However, in most cases these specific modeling tools lack primitives that allow the consideration of dynamic aspects. Furthermore, they do not provide facilities to take elements of the domain environment into account.

Knowledge representation systems in general are difficult to use and require a good background in knowledge representation techniques. Their advantage is that they provide what has been called "intelligent support" such as semantic retrieval and consistency checking. From the different common Knowledge Representation systems only the so-called object-centered systems (semantic network systems, inheritance hierarchies, frame systems) or a hybrid-one that includes an object-centered component seem to be appropriate for our modeling concern.

Hypertext systems are natural and easy to use. Building a hypertext network is a kind of informal knowledge engineering. The difference to knowledge representation is that the goal of the hypertext user is often to capture an interwoven collection of ideas without regard to their machine interpretability. Thus hypertext systems do not provide any "intelligent support" but they provide powerful mechanized operations and processes to build, manipulate, and view a "networked" user-interface.

Special modeling languages for software engineering such as GIST (Balzer 1981; Goldman, and Wile 1980), or REFINE (Smith, Kotik, and Westfold 1985; Goldberg 1986), are targeted on requirements engineering, systems specification, or prototyping. Despite that difference in the underlying goal, they are interesting within the context of domain analysis for GenSIF. Their mostly rich set of different modeling primitives generally has an underlying object-based approach with some specific other constructs. These other constructs allow to capture information concerning

dynamic aspects as well.

Object-oriented analysis and design systems, as they are described, can handle static aspects, dynamic aspects and even communication within an application domain. However they are oriented towards specific application projects. Furthermore they do not provide facilities for reasoning and the final model cannot be regarded as human oriented.

To give an answer to the question which of the briefly discussed candidate systems is the best choice regarding its application as domain analysis support tool for GenSIF requires that practical tests are carried out.

8.3 Conclusions

Domain analysis is an important concept of the GenSIF framework. It provides the other GenSIF components and activities with a model of the application domain. The role of the provided domain model in these steps is of high importance, since what is done presupposes the existence of an adequate domain model.

Domain analysis in general is a young research area and the opinions about the concrete contents of the domain analysis process vary sometimes to a large degree. This is the direct implication of the fact that domain analysis always is oriented towards a specific goal, which in our case is systems integration.

One major finding was that in order to facilitate the selection or design of the fitting integration architecture, a conceptual analysis regarding the application domain under consideration of specific aspects has to be performed. Since the integration architecture must also fit to the dynamics within a domain and the communication that takes place in a domain, these aspects must be considered in the domain analysis and therefore in the resulting domain model as well.

In comparison to other domain modeling approaches, it can be said that the domain modeling approach within GenSIF strives to build a much more compre-

hensive model of the real world but neglects any constructive or design approach as part of the domain modeling process.

The developed approach to a domain modeling notation for the specific needs of GenSIF at the current status can only be regarded as a first attempt towards such a specific notation. However it can help to discuss domain analysis and domain modeling within the GenSIF framework on a more concrete level.

A similar notation could be used as the "heart" of a research tool implementation of a specific domain analysis support tool for GenSIF. To talk about implementation issues of such a notation does not just mean to think about the possibilities to transform the representation structures introduced in chapter 6 into a machine processable form. The main point is that we would like to have the semantics of those representation structures as much as possible "understood" and interpretated by the computer.

What is required as tool for domain analysis with respect to GenSIF is not just any tool that facilitates modeling of an application domain. This is just the basic purpose such a tool must facilitate and the set of modeling primitives of the underlying modeling formalism is only one aspect to consider. Beyond that, additional aspects to consider are:

- the level of automatic support for designing, manipulating and utilizing the domain model
- the presentation of the domain model to the user.

An analogy can be drawn to the criterias one would apply to select a word processing system which best fits his/her needs. Besides others, one criteria certainly would be the level of automatic support provided by the system to create efficiently documents of high quality. Another criteria would be concerned with the way a document would be presented to the user by the system.

As discussed in section 7.3 the level of automatic support provided by such a tool basically depends on the computational processability of the domain model, which can cover the syntax only or some semantics of the model as well. Here the "ultimate objective" could be such a high level processability of the domain model that it could be used by an "automatic generator" of integration architectures and by an "automatic requirements analysis assistant". This would lead to something that could be called "Computer Aided System Engineering and System Development with GenSIF".

From the authors perspective and to the current status, the final choice regarding the selection of a domain analysis support tool for GenSIF tends to be between knowledge representation systems and hypertext systems. However, the final decision depends on aspects that have not been fixed yet. These aspects are related to the question if we should favor a domain model that allows to include more "automatization" within the GenSIF framework later on. In this case our choice should be a knowledge representation system since the necessary computational processability with respect to the semantics of the domain model is facilitated by such a system.

The other alternative is to favor a more natural and convenient to use domain analysis support tool that emphasises human understandability. In this case the choice would be a hypertext system. However it has to be accepted that this decision implies a domain model where the contents stays uninterpreted. Hence, this alternative does not offer any potential for "automatization" within GenSIF based on the domain model.

This thesis does not provide something like a complete "recipe" for domain analysis as concept of GenSIF. To keep once more this metaphorical "language", it is more concerned with the "central ingredients" of it, namely its domain modeling component and the kind of support tool that is appropriate to it. However, the given
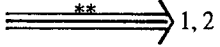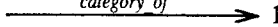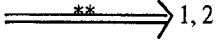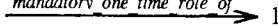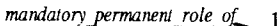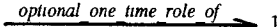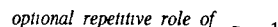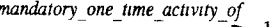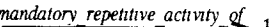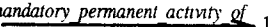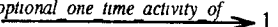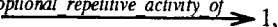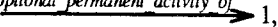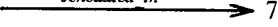
results are an advanced basis for the continuation of research within that area.

The author sees two independent approaches to continue with his research. One more long term oriented approach would be the development, formalization and implementation of a specific modeling language for GenSIF. The presented approach to a specific modeling notation for GenSIF might be a good "platform" for that continuation. A more short term oriented continuation would be the evaluation of concrete available tools with respect to their applicability to domain analysis as concept of GenSIF.

APPENDIX

Overview of Proposed Modeling Primitives:

Node Primitive / Properties	Diagrammatic Symbol	Applicable Link-Primitives	
		partially predefined	completely predefined
1 Object Class	(hexagon) **	Object-Class-Relationship. ⟹ ** 1, 2	_category of_ → 1
2 Role	(trapezoid) *	Role-Relationship: ⟹ ** 1, 2	_mandatory one time role of_ → 1 _mandatory repetitive role of_ → 1 _mandatory permanent role of_ → 1 _optional one time role of_ → 1 _optional repetitive role of_ → 1 _optional permanent role of_ → 1
3 Activity	(parallelogram) *		_mandatory one time activity of_ → 1, 2 _mandatory repetitive activity of_ → 1, 2 _mandatory permanent activity of_ → 1, 2 _optional one time activity of_ → 1, 2 _optional repetitive activity of_ → 1, 2 _optional permanent activity of_ → 1, 2 _scheduled in_ → 7

90

Node Primitive	Diagrammatic Symbol	Applicable Link-Primitives	
		partially predefined	completely predefined
4 Communication Element	*		*sent by* → 1, 2, 5 *received by* → 1, 2, 5 *implied by* → 3, 6
5 Environmental Object	**	Environmental-Participation -Relationship: - - - **- - - → 3	
6 Environmental Activity	*		*activity of* → 5 *scheduled in* → 7
7 Time Frame	*		*is time frame in* → 7

* : location of identifier

** : location of identifier (all in capital letters)

comment: The number(s) at the end of each arrow denote(s) the ID of node-primitives the link can be connected to.

REFERENCES

Allen, J. 1981. "A General Model of Action and Time." *Technical Report 76, Dept. of Computer Science, Rochester University*

Bachman, C.W. 1977. "The Role Concept in Data Models." *Proc. 3rd Int. Conf. on Very Large Databases, Tokyo, Japan*

Balour, A., Anderson, T., Dekeyser, J., Wong, H.K.T. 1981. "The Role of Time in Information Processing: A Survey." *ACM SIGMOD Letters* April

Balzer, R.M. 1981. "A Summary of Gist." *Technical Report, USC Information Science Institute*

Barstow, D. 1985. "Domain-specific Automatic Programming." *IEEE Transactions on Software Engineering.* SE-11(11): pp. 1321-1336

Booch, G. 1987. "Software Engineering with Ada." Benjamin Cummings, Redwood City, CA, USA

Borgida, A., Greenspan, S., Mylopoulos, J. 1985. "Knowledge Representation as the Basis for Requirements Specification." *IEEE Computer* April: pp. 82-91

Brodie, M. 1982. "On the Development of Data Models." in Brodie, Mylopoulos, and Schmidt 1986, pp. 19-47

Brodie, M., Mylopoulos, J., Schmidt, J. 1986. "On Conceptual Modeling." Springer-Verlag

Bubenko, J. 1983. (ed) "Infomation Modeling." Studentlitteratur, Chartwell-Bratt Ltd

—. 1980. "Information Modeling in the Context of System Development." *Proceedings of IFIP 80* pp. 395-411

Carasik, R.P., Johnson, S.M., Patterson, D.A., von Glahn, G.A. 1990. "Towards a Domain Description Grammar: An Application of Linguistic Semantics." *ACM SIGSOFT, Software Engineering Notes* vol. 15, no. 5: pp. 28-43

Conklin, E.J., 1987. "Introducing Hypertext." *IEEE Computer* Sept.: pp. 22-41

Devanbu, P., Brachman, R.J., Selfridge, P.G., Ballard, B.W. 1990. "LaSSIE: a Knowledge-based Software Information System." *Proc. 12th Int. Conf. on Software Engineering* pp. 249-261

Prieto-Diaz, R. 1990. "Domain Analysis. An Introduction." *ACM SIGSOFT, Software Engineering Notes* vol. 15, no. 2: pp. 47-54

Prieto-Diaz, R., Arango, G.F. 1991. "Domain Analysis and Software Systems Modeling." IEEE Computer Society Press, pp. 9-32

Dubois, E., Hagelstein, J., Lahou, E., Ponsaert, F., Rifaut, A. 1986. "A Knowledge Representation Language for Requirements Engineering." *Proceedings of the IEEE* vol. 74, no. 10: pp. 1431-1444

Dubois, E., Hagelstein, J., Lahou, E., Rifaut, A., Williams, F. 1986. "A Data Model for Requirements Analysis." *Proceedings of the 4th Conference on Data Engineering* pp. 646-653

Embley, D.W., Kurtz, B.D., Woodfield, S.N. 1992. "Object- Oriented Systems Analysis." Yourdan Press, Englewood Cliffs, NJ, USA

Fickas, S. 1987. "Automating the Analysis Process: An Example." *Proceedings of the 4th Int. Workshop on Software Specification and Design*

Garg, P.K. 1988. "Abstraction Mechanisms in Hypertext." *Communications of the ACM* vol. 31, no. 7: pp. 862-879

Greenspan, S.J., Mylopoulos, J., Borgida, A. 1982. "Capturing More World Knowledge in the Requirements Specification." *Proc. 6th Int. Conf. on Software Engineering* pp. 225-234

Greenspan, S.J. 1984. "Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definitions." *PhD Thesis, University of Toronto, Technical Report CSPG-155*

Goldberg, A.T. 1986. "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques." *IEEE Transactions on Software Engineering* SE12: pp. 752-768

Goldman, N.M., Wile, D.S. 1980. "A Database Foundation for Process Specifications." *Technical Report ISI/RR-80-84, USC Information Science Institute*

Iscoe, N., Liu, Z.Y., Tam, K.Y. 1991. "A Framework for Understanding and Discussing Domain Modeling." *Proc. 13th Int. Conf. on Software Engineering, Domain Modeling Workshop, Austin, Texas, USA* pp. 5-13

Iscoe, N., Williams, G.B., Arango, G. 1991. "Domain Modeling for Software Engineering." *Proc. 13th Int. Conf. on Software Engineering, Domain Modeling Workshop, Austin, Texas, USA* pp. 1-4

Jackson, M. 1983. "System Development." Prentice-Hall International

Kangassalo, H. 1983. "Structuring Principles of Conceptual Schemas and Conceptual Models." in Bubenko 1983, pp. 223-307

Kelly, V.E. 1991. "The KITSS Project: Domain Modeling to Support Functional Testing of Evolving Embedded Software Systems." *Proc. 13th Int. Conf. on Software Engineering, Domain Modeling Workshop, Austin, Texas, USA* pp. 109-113

Lubars, M.D. 1988. "A Domain Modeling Representation." *MCC Technical Report Number STP-366-88*

Mylopoulos, J., Borgida, A., Greenspan, S., Wong, H. 1984. "Information System Design at the Conceptual Level - The Taxis Project." *Database Engineering* vol. 3: pp. 185-190

Neighbors, J. 1981. "Software Construction Using Components." *PhD Thesis, Department of Information and Computer Science, University of California, Irvine*

—. 1984. "The Draco approach to constructing software from reusable components." *IEEE Transactions on Software Engineering* SE-10(9): pp. 1247-1267

Olle, W., Sol, G., Verrijin-Stuart, A. 1982. "Information Systems Design Methodologies. A Comperative Review." North-Holland Publishing Company, Amsterdam

Ross, D.T 1977. "Structured Analysis (SA): A language for communicating ideas." *IEEE Transactions on Software Enginerring* SE-3(1): pp. 16-34

Rossak, W., Ng, P.A. 1991. "Some Thoughts on Systems Integration: A Conceptual Framework." *International Journal of Systems Integration* vol. 1, no. 1: pp. 97-114

Rossak, W., Prasad, S. 1991. "Integration Architectures - A Framework for Systems Integration Decisions." *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics, Charlottesville, VA, USA* pp. 545-550

Rossak, W. 1991. "Domain Modeling and Systems Integration." *Proc. 13th Int. Conf. on Software Engineering, Domain Modeling Workshop, Austin, Texas, USA* pp. 150-153

—. 1992a. "Integration Architectures, A Concept and a Tool to Support Integrated Systems Development." *Internal Report, Department of Computer and Information Science, New Jersey Institute of Technology*

—. 1992b. Presentation of the GenSIF framework in the first seminar for systems integration at New Jersey Insitute of Technology (spring term).

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

Rich, C., Waters, R., Reubenstein, H. 1987. "Toward a Requirements Apprentice." *Proceedings 4th International Workshop on Software Specification and Design*

Setliff, D.E. 1991. "Domain Modeling in a Real-Time Software Synthesis System." *Proc. 13th Int. Conf. on Software Engineering, Domain Modeling Workshop, Austin, Texas, USA* pp. 162-167

Smith, D.R., Kotik, G.B., and Westfold, S.J. 1985. "Research on knowledge-based software environments at Kestrel Institute." *IEEE Transactions on Software Engineering* SE-11(11)

Sowa, J.F. 1984. "Conceptual Structures, Information Processing in Mind and Machine." Addison-Wesley Publishing Company

Zemel, T. 1992. "MegSDF - Mega-Systems Development Framework" *PhD Thesis Proposal, Department of Computer and Information Science, New Jersey Institute of Technology, June 1992*