Deliverable 3

Specification of the HIDE Environment

Esprit Project 27439 - HIDE

High-level Indegrated Design Environment for Dependability

A. Borschet, M. Dal Cin, J. Jávorsky, A. Pataricza, G. Savoia, Cs. Szász

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Conzorcio Pisa Ricerche-Pisa Dependable Computing Centre (PDCC)
Technical University of Budapest (TUB)
MID GmbH
Intecs Sistemi SpA

HIDE/D3/TUB/1/v2

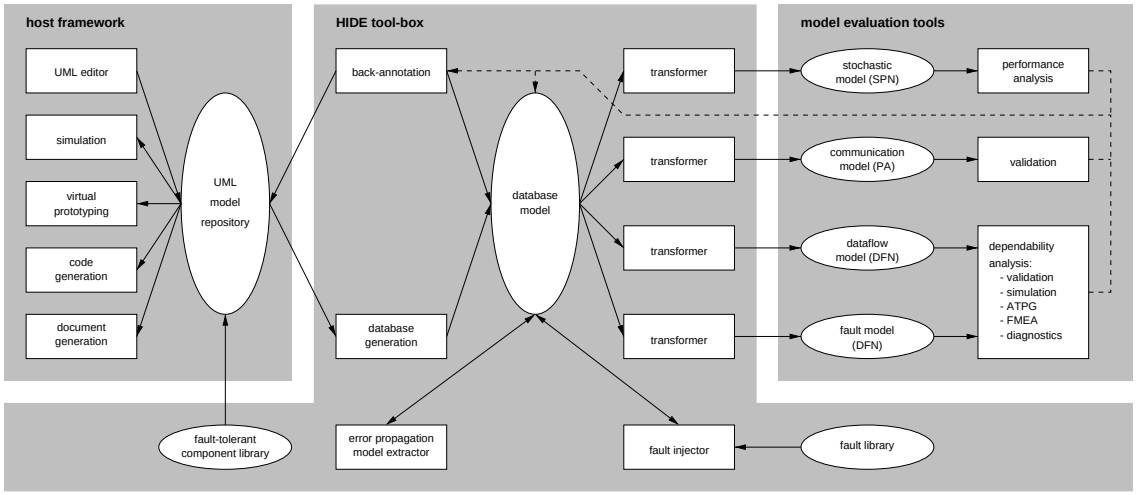

Figure 1: Architectural framework of the HIDE environment

1 Definition of architectural framework

The aim of this deliverable is the definition of the HIDE architecture in a form which can be used without major alteration for both phases. However, the implementation could differ for the two phases. The primary target of Phase 1. is the definition of a prototyping environment, in consideration of the assurance of a high level of flexibility and of a good support for debugging of the algorithms to be implemented. In this phase, both the efficient use of the resources and the time requirements of the transformations are of secondary importance. Accordingly, this implementation should rely, as far as possible, on commercially available tools, not necessary incorporated into the final HIDE tool.

The HIDE architecture is built up from 4 main components:

UML CASE tool The user-end modelling platform is an arbitrary UML CASE tool. In this component the user can build up its own UML model. All tool provided features, as code-generation, round trip engineering, can be used freely, without any modification. When modelling new constraints and rules must observed which are defined in the HIDE environment. As the CASE tool is one of the mostly used user-end interface of the HIDE framework, if definitely required, it may be altered in some way to suit into the requirements raised by HIDE.

Verification tools These tools usually are off-the-shelf components and are not supposed to be altered in any way. The HIDE core will act as an end-user toward the verification tool. It will call the tool supplying the correct input for it and then will analyze the result of it. As the development or improving of these tools are totally out of scope of the HIDE aims, their knowledge and shortcomings have a great influence on the services delivered by HIDE, in this sense we do not alter the tools, but merely support the selection of proper modelling paradigms and tools for the problem solving.

HIDE repository All additional functions added to the CASE tool are developed on the basis of a virtual database system. This has many advantages presented in later sections, but the most important three are:

- independence of CASE tool
- better performance, then reading and writing the internal repository of the host CASE tool
- faster development as all model transformations are implemented on a uniform environment

The HIDE repository will contain one or more actual UML models (designed in the CASE tool), the functions executed on it and will act as a fault tolerant library.

The use of a standard database for implementing the HIDE reposotory offers naturally all services provided by the database system. Such services can provide additional benefits for the user. For instance, here are some useful possibilities:

- Availability of the UML model through a standard interface (SQL). This way the user can utilize the power of the database based approach.
- If the CASE tool doesn't provide support for version checking or automatic backup, the end user could use the database for these tasks.

Naturally the disadvantage of this approach is that, as a matter of fact, there is a duplication of databases, one is the CASE tool's UML model repository, the second one is the HIDE repository. It has to be pointed out, that:

The majority of the tools do not use standard databases, however according to the needs of the UML based design process (like the support of teamwork, specialized data formats, ...) they offer some very specialized solutions. The next problem in merging the repository and database would be the necessity of a common script language. Currently different tools use different script languages, for instance Innovator uses Tcl/Tk, Rational Rose uses Basic Script, UML Nice provides a CORBA interface. Specialized database script languages are provided by the tools vendors for handling the database contents, in the first phase we always relied on the PL/SQL script language which is an standard provided by practically all database vendors. An additional benefit of using the standard databases is, that this way the compliance with the UML metamodel is assured by automatically transforming the meta-model of UML into database structure. This meta-model was made available by the OMG for end-users, and as a matter of fact by using the advanced features of CASE tools supporting the transformation of a UML model into a database structure as well, a fully automatic transformation procedure was supported, automatically assuring the standard compliance. This approach is at the same time a well founded basis for the maintenance of the entire HIDE approach as if further modifications are taken on UML, only some minor changes are needed in the scripts interfacing the automatically derived database structure from the UML meta-model.

all UML features which are supported and used by the HIDE framework
extensibility features for the end-user: internal script language
several requirements on the script language

Table 1: Requirements for UML CASE tools

HIDE toolbox. The HIDE toolbox will be a user-end interface executing those functions that cannot be triggered from the CASE tools. For example these are the database manipulation tasks.

1.1 Objectives

One of the main requirements when defining the architectural framework is the openness of the approach. On the openness we mean that:

- an arbitrary UML CASE tool can be used as host environment assumed that it is able to export the model for further modification. Here the output format of the model repository has to be well defined. Such definitions are available for the majority of the tools, like UML Nice, Innovator, Rational Rose. This kind of extensibility is naturally only the simplest solution for integrating the HIDE environment into a new host framework. Typically tool providers support the integration of user functions in two ways: by an internal script language or by a CORBA interface in order to make the content of the repository reachable for user defined tools, like HIDE.
- The user interface of the host CASE tool has to be extended (for example, by new menus) in order to serve the comfort of the use of the HIDE environment.
- There should be some kind of back-annotation mechanism in order to provide the possibility for the visualization of the analysis results provided by the verification tools.

1.2 CASE tools

As the CASE tool is one of the most basic resources for the entire HIDE approach, not only in the sense that it provides the user front-end interface, but merely that it has to integrate the additional services offered by HIDE, it may be altered or extended by the features required by HIDE.

An arbitrary UML CASE tool will play the part of the user-end interface of the HIDE framework. The user will use this component for design an UML model, the subject of all later investigations performed by the HIDE engine. Such a tool, in order to enable its integration into the HIDE framework, must provide – in addition to all the standard features specified in the UML – certain necessary features. An overview on the requirements for a CASE tool is given in Table 1 and clarified in full details in Section X.

The integration into the HIDE will extend the tool with the following features:

- Transformation (export) of the UML model designed in the CASE tool into the HIDE repository.
- Transformation (import) of the HIDE repository into the tool. This is required when the UML model is modified inside the HIDE repository.
- Visualization of the results of the verification tools.

1.3 Analysis tools

The HIDE framework is supposed to use and integrate several analysis tools. The main innovation in HIDE is to provide the knowhow necessary for problem solution in the form of model libraries and modelling paradigms. This additional knowhow will be formulated in such a way which is fully compliant to the actual analysis tool. These tools will perform several verification / validation tasks on the investigated UML model. They will enable the end-user to identify design errors in the model or derive qualitative or quantitative parameters for the model.

As the end-users of the HIDE framework are not supposed to know neither the integrated verification tools, nor their underlying mathematical theory, the framework must "hide" these tools in such a way that the user can start the verification directly from the front-end and receive only the post-interpreted results of the output generated by the tools.

These tools generally are not designed in any way for analyzing an UML model. Because of this the conversion between them and the UML model must be solved in some way. There are two possibilities to overcome the gap between the tool's internal representation and the UML model to be analyzed.

- First is to extend the verification tool with new capabilities that enable them to handle UML models directly. As the HIDE doesn't aim the development of any new tool, but at the reuse of them, these tools will be off-the-shelf components of the HIDE framework. This way, if they are unprepared to handle UML models or they do not provide any extension interface for the developer, the only solution is to accept the tool as it is.
- The second solution, which is one of the basic concepts of HIDE, is to provide correct input for the tool, which corresponds to internal representation and to the syntax of their input files. Naturally, this solution requires as starting point the precise syntax of the input files.

1.4 HIDE repository

The HIDE repository is the core component of the HIDE framework. It is responsible for the following three main tasks:

- Store one or several UML models designed with an arbitrary UML CASE tool. For this purpose a complex database scheme is built up which correspond to the official UML meta-model published by the OMG. In order to guarantee that the HIDE framework is

independent of the CASE tool, this scheme is not supposed to differ from the standard meta-model except if it is definitely necessary. As a result of this independency the HIDE environment doesn't utilize the tool specific features even if they would improve somehow the framework.

- Optionally store one or several models for target verification tools. As the UML models are stored in a UML database scheme, the models for the target tools (optionally) could also have a database scheme which could act as an intermediate storage place when transforming the UML models for the target tools. This could be very useful when the transformation between the UML model and the target model is too complex to handle every aspect of the transformation in one step.

- Store and perform the transformation procedures executed between the UML model and target tools and vice-versa. The functions performing the transformations between different models can be stored in the database or in the local file-system. They can be executed from the CASE tool or from the HIDE tool box. In the first case the whole HIDE framework can be hidden from the end-user: the model design, the model analysis, the interpretation of the results can be performed through the CASE tool.

- Store the HIDE fault-tolerant component library.

- Store and perform the model manipulation procedures (e.g. fault injection, modification for fault-tolerance) taking as input a UML model and the HIDE component library, while the output is the modified UML model.

In the definition of the HIDE architecture we define only a "virtual" HIDE database, without specifying its implementation. It is obvious that a storing device is needed for storing the fault tolerant component library. In the following several alternatives of implementing the HIDE repository are sketched out:

- The storage methods are implemented manually. This means storing the data in individual files, creating thus a new file format: the HIDE format.

- Using the front-end CASE tool as a storing device, as the data to be stored is very close to the UML, majority of them just UML components, like classes, attributes, methods.

- Using a commercial database system. This way we get a powerful storing device and a CASE tool independent development platform and language. As this solution is used in the first phase this will be presented in full detail in Section 2.1.

- There is a variety of possible implementation of the HIDE repository. In the following sections some standard sollutions will be presented, as UREP, XMI and XML.

1.4.1 Universal Repository (UREP)

The Universal Repository (UREP) is a dynamic, extensible information system that defines, integrates and manages metadata and business data. It is based on leading industry standards that support best of breed tool integration and interoperability across heterogeneous platforms. UREP is being used by independent software vendors (ISVs) and large corporations in diverse domains such as:

- Tool Interoperability
- Enterprise Asset Management
- Legacy Integration
- Application Life Cycle Management
- Component-based Development
- Corporate Meta Data Management
- Data Warehousing

1.4.2 XMI and XML

The XMI specification allows developers to create distributed applications in a vendor-neutral environment, and demonstrates the commitments of IBM, Unisys and Oracle to providing standards-based technology to the development community. XMI aims to make the eXtensible Markup Language (XML) – integrated with the OMG's Unified Modeling Language (UML) and Meta Object Facility (MOF) – the cornerstone of an open information interchange model. These standards are already in use by many major software vendors.

XML is the recommendation set forth by the World Wide Web Consortium (W3C) for defining, validating and sharing document formats on the Web, while UML is designed to give application developers a common language for specifying, constructing and documenting distributed objects and business models. MOF is an OMG standard for distributed repositories and metadata management.

"The rapid integration of modeling and repository technologies from the OMG and W3C into the XMI specification is a great example of industry consensus on how metadata practically unifies diverse technologies. As today's demonstrations indicate, our vision – that developers would visually design models for various domains and then share objects and metadata regardless of development tool or environment – is quickly becoming reality."[1]

Today's demonstrations by Unisys, IBM, Oracle and SELECT Software Tools illustrate the practical value of XMI. The vendors showed the exchange of UML model between a variety of modeling and development tools and repositories, including IBM's VisualAge for Java development environment, IBM VisualAge TeamConnection enterprise repository, the object building

[1] Sridhar Iyengar, Unisys Fellow and chair, OMG Object Analysis and Design Task Force

technology of IBM WebSphere Enterprise Application Server, IBM DB2 Universal Database, Rational Rose, SELECT Component Factory (SCF), Unisys UREP repository, Oracle Repository and Oracle Database Designer. Each modeling tool is used to view and extend the model before passing it on to the next vendor.

1.5 Fault-tolerant components library

The extension of a UML CASE tool with fault-tolerant features requires several predefined fault-tolerant components. The end-user will be able to extend (or convert somehow) it's UML model with these predefined components without an in-depth knowledge of their internal function. As the HIDE framework should be independent of any CASE tool, these components must be stored in the HIDE repository.

There are two possibilities for extending a UML model with fault-tolerant components:

- Uploading directly (into the host CASE tool) the selected components from the fault-tolerant library stored in the HIDE repository.
- Exporting the UML model into the HIDE repository, manual (with help of the HIDE toolbox) or automatic extension of this model with the needed components, re-importing the extended UML model into the CASE tool.

1.6 The back-annotation mechanism

In work Phase 1. no back-annotation mechanisms are planned, but the implemented transformations try to propagate as many information about the original UML model as possible. This is necessary, when we want to understand, moreover, to automatically analyze the results generated by the analysis tool. The easiest way for enabling back-annotation is by preserving the internal IDs given by UML CASE tool. This way, we can point directly to the element in the original UML model in the host CASE tool.

In Phase 2, when implementing the transformations, additional effort must be addressed for preserving the information (at least the IDs of the elements) needed for back-annotation.

2 Specification of HIDE components

This section contains the selected and implemented solutions of the HIDE Phase 1. These solutions are supposed to be the basic solutions for Phase 2.

In this approach the main idea is the use of a database system as a basis, on the top of which all HIDE functions can be built.

The advantages of this solution are the following:

- Easy storage of HIDE persistent data, like fault-tolerant component library.

- All HIDE functions – like model transformations, model enrichment, model manipulation – can be implemented upon the same platform and independent of the UML CASE tool. This way the switch from one CASE tool to another will not affect the core of the HIDE environment (the implementation of the model-transformations). These parts are the functions implementing the data-exchange between the CASE tool and the HIDE UML database. It must be pointed out again, that the real knowhow of the HIDE implemented using this solution is independent of the CASE tool, so its knowhow is accessible to every CASE tool provider after the transformation and back-annotation between them and the UML database is implemented. The effort needed for implementing this part depends on the level of compatibility between the CASE tool's internal repository and the standard UML meta-model. Although, on the market today there is no UML CASE tool supporting all UML standard features, it is a natural expectation that the UML CASE tools will move toward a better compatibility in the future. Moreover, there will be some tools, which will use exactly the UML meta-model as the basic when developing their internal meta-model and storage architecture.

- Rapid prototyping: The well defined standard and power of the DML [2] languages of the database system enables not only a rapid development for model transformation but also a very effective one. In the case of HIDE complex model transformation are needed – for example between a UML model and a Petri Net – and the input models are huge and complex ones. When manipulating big data it is straightforward to use a database system, which can provide easy and effective access to the data.

- Platform independence: In the case of HIDE it can occur that the CASE tools and the analysis tools are executed on different platforms. As most database systems are reachable from various platforms, this "platform gap" can be bridged through the database.

The drawbacks of this approach are the following:

- The need of a commercial database system, which makes the HIDE more expensive.

- The UML model to be analyzed is duplicated: it is stored in the host CASE tool and in the database as well.

- Consistency problems must be resolved between the two copies of the model.

- A data-exchange feature must be developed between the CASE tool and the database (however it is a quite mechanical task).

- The data-exchange decreases the overall performance of the HIDE framework (however it is very likely, that the performance improvements gained in the case of the model-transformations executed inside the database, will abundantly compensate it).

[2] *Data Manipulation Language*: script language for selecting, inserting, deleting and upgrading database tables

2.1 HIDE repository

Main tasks to provide:

- Storage of UML models designed in a CASE tool
- Storage of the generated models for target tools
- Storage and execution of the transformation library (written in PL/SQL)
- Storage of fault tolerant components (conform to the UML meta-model).

In Section 1 the HIDE repository was defined as a "virtual" HIDE database without specifying its implementation. In Phase 1., however, several features were implemented, and these required an implementation of the HIDE repository.

The most important features required on the side of the database are enumerated in the following:

- Required features:
  - Relational database: the database scheme generated from the UML meta-model can be used only for relational database systems.
  - Client-server architecture: as the database must be reached from different tools.
  - Reachable from different platforms: the tools can be of different platforms.
  - PL/SQL as internal DML language: the model transformation algorithms are implemented in PL/SQL in Phase 1.
- Optional features:
  - Reachability through ODBC interface: for online database connection from the CASE tool.
  - Storage of PL/SQL program packages: the PL/SQL procedures can be stored in the database, thus we can reach all HIDE features through only one interface, which is the database.
  - Server side execution of PL/SQL packages: better performance can be obtained when the features manipulating the database is executed by the database server.
  - Hierarchical queries: as the UML has many recursive stuctures (for example: states, which have substates, which have substates, ...) the use of hierarchical queries can have a major impact on the performance.

Because of its advantages presented in the previous section, we choosed the database system based solution, more precisely, we implemented the HIDE repository on the base of the relational database system provided by ORACLE Corporation.

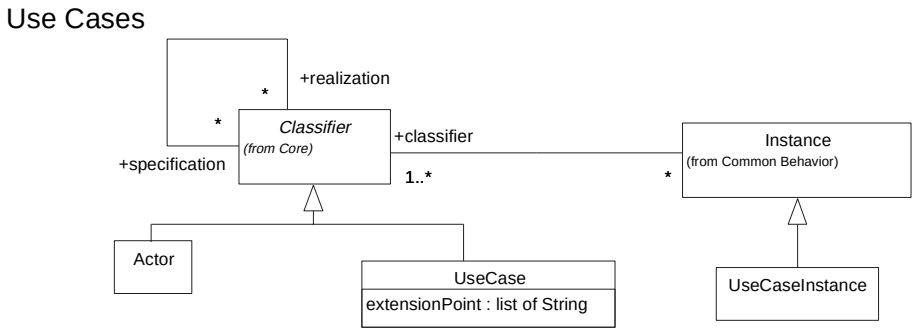

Figure 2: Meta-model of the Use Cases in the UML provided by Rational Rose

2.1.1 Database representation of the UML meta-model

The first task of the HIDE repository is to store an arbitrary UML model designed in a CASE tool. The structure of the database scheme, which will hold these models, was a principle issue when implemenating of the HIDE repository. There were two possibilities: to mirror the database of the CASE tool or to develop a tool independent and based fully on the UML standard meta-model provided by OMG. As even in the phase 1. two CASE tools (Innovator and UML Nice) were involved and as the UML meta-model is downward compatible with the internal representation of these tools, the second approach was the most suitable, which is to generate a database scheme from the UML meta-model.

The UML meta-model, published by OMG, was provided publicly and downloadable by the Rational Software Corporation (Figure 2). The UML meta-model was specified in UML itself, using the format of Rational Rose. Task to be solved was to generate the database structure by a DDL[3] script. Although DDL scripts can be generated from UML models directly, this part of the UML CASE tools are inefficient to correctly handle such big models. We choosed to convert the UML model to ER[4] model first, from which the DDL generation has a standard theoretical basis.

As the main front-end CASE tool (Innovator) integrated in the HIDE in Phase 1 has a ER module as well, we used this tool for generating the DDL scripts.

The steps for generating the DDLs for the UML meta-model are the following:

- Conversion the Rose model into an Innovator UML model. This could be resolved easily as Innovator can be extended to understand the file formatof the Rose.
- Conversion of the UML model into an ER model. Innovator has a feature which can convert an UML model to an ER model. When doing such a conversion the conceptual differences had to be resolved. Some of the were solved automatically, while others were resolved manually.
- Generation of the DDL scripts for creating the HIDE UML database scheme. These scripts

[3] *Data Definition Language*: DDL commands include commands to creating and altering databases and tables.
[4] *Entity Relationship model*: modelling paradigm for designing relational databases

can be generated automatically from Innovator.

This way, if in the future the standard UML meta-model is changed, by redoing these three steps, we get nearly automatically a new, upgraded UML standard database scheme. Of course all transformations which use this scheme somehow must be upgraded as well.

2.1.2 Transformation library

The transformation library has to store all functions which will perform complex model transformations. The majority of these transformations have as source an UML model and as target the model of an analysis tool. The output of these transformations can be:

- The input file of the target tool. In this case, both the model transformation and the file export conform to the target tool's file format, must be solved in one step. The implementation of these orthogonal functions in one monolithic function, makes the resulting code less maintainable.
- A new model stored in the database scheme built up to hold the models of the target tools. This suppose that the target database scheme was previously designed and built up and that there is a procedure which will export this model for the target tool. In spite that this technology requires two more steps in contrast with the previous solution, this solution is more efficient as the development steps and their tasks can be partitioned more easily. The biggest advantage is that the most complex part, which is the transformation between the source and target model, can be implemented and executed on the basis of a database exploiting all its power. Thus, the complex transformations can be implemented completely in the script (DML) language of relational databases (PL/SQL), without caring about non theoretical aspects of the transformation, like generating codefor the target tool.

Thus, here are some of the advantages, of this approach:

Easy development Every transformation can be implemented in the same language, independently of the source and target tools.

Efficiency and performance In DML languages the input-output operations are very effective and easy to use. These model transformations need many of these operations, so the overall performance of developing such functions can be improved a lot.

Execution by the database server If the data manipulations are executed by the database server, then by reducing the communication overhead and exploiting the automatic optimization performed by the server, the execution of the entire function will be speed up.

Client-server architecture As the database server can be connected from client computers and the transformation scripts are executed on the server, the overall performance of the transformations depends mostly on the server, which is usually a powerful machine.

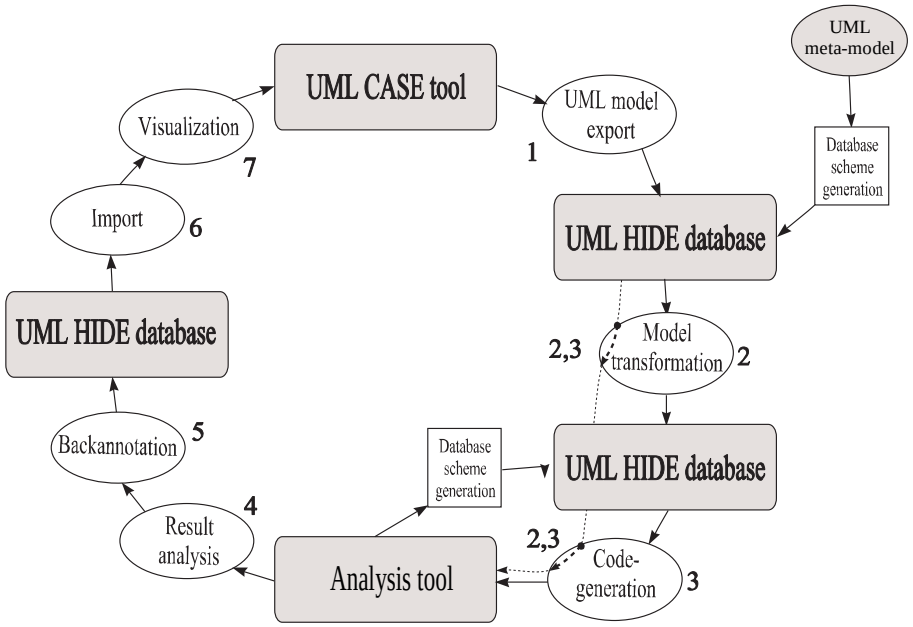

Figure 3: Analysis of an UML model

Thus, the function performing the complete model-transformation from the UML model to the analysis tool is partitioned into three sequentially executed parts (Figure 3, first three steps):

1. Export of a UML model into the HIDE UML database.
2. Model transformation between the source UML model and the target model inside the database.
3. Exporting the generated target model for the target tool conform the syntax of it's input file.

The next step od the model-transformation is the execution of the target tool using the generated file as input. Then, the result must be processed and visualized somehow for the end-user, as the results are probably difficult to understand.

Thus, the steps of a model transformation and analysis (Figure 3, last four steps):

1. Implementation of the transformation between the source UML scheme (holding a UML model designed in the host CASE tool) and the target tool.
2. Implementation of the back-annotation of the result generated by the tool into the source UML model.
3. Re-import of the modified (by the back-annotation) UML model into the CASE tool and visualization of the results.

2.1.3 Database scheme of the target tools

As Figure 3 shows there are two possibilities to transform a UML model from the database to a representation of the target analysis tool (This task is represented by the 2^{th} and 3^{rd} step in the Figure.):

- One step solution: performing the model-transformation and the code-generation in one step. The effectiveness of this solution depends on the complexity of the target model.
- Two step solution:

1. Transformation from a UML model (database scheme) to a target tool's model (database scheme).
2. Code generation (input file) for the target tool from its representation in the database.

   This simplifies the model-transformation, because it is partitioned into two orthogonal parts.

Development steps for an arbitrary code-generator for a target tool:

1. Building up in UML or in ER paradigm the internal meta-model of the target tool.
2. Generation of the database scheme for the target tool.
3. Implementation of the transformation between the source UML scheme and the target tool's scheme.
4. Implementation of the code-generator from the tool's scheme to the target tool.
5. Implementation of the processing of the result generated by the tool.
6. Optional: Back-annotation of the results into the source UML model.

2.2 CASE tools

Needed extensions:

- Export of a UML model into the database (on-line, off-line) (implemented).
- Import of a UML model from the database (not implemented).

2.2.1 Innovator

Innovator overview

Provided by	MID GmbH
Used in first phase	Yes
Supported platforms	Windows 95 and Windows NT, OS/2, UNIX (AIX, Solaris, HP-UX, SINIX), AlphaVMS
Script language	Tcl/Tk

The concept Innovator is a global CASE solution for analyzing and designing IT systems. The modular concept of the Innovator workbenches allows horizontal and vertical method integration for a seamless transition all the way from business process engineering to system integration.

The Innovator workbenches are based on the standards and methods of function-oriented, data-oriented and object-oriented software development.

Innovator is successfully used by over 13,200 users for application development. With a variety of features, the Innovator design supports work in large project groups.

The following notations are supported:

Business Process Engineering:

- Unified Modeling Language (UML)

Object Orientation:

- Unified Modeling Language (UML)
- Object Modeling Technique (OMT)

Data Modeling:

- Entity Relationship Modeling (ERM)
- Structured Entity Relationship Modeling (SERM)

Function Orientation:

- Structured Analysis (SA)
- with Real-time Extension (RT)
- Structured Design (SD)
- Modular Design (MD)

Special Features:

- Mapping OO-ER
- Package Diagrams

Client/Server Innovator is designed as a true client/server system. All data is collected in data pools, or "repositories", which the clients do not have direct access to. The management of a repository is the responsibility of a server program alone. The server ensures:

- the consistency of the managed data within the projects and models
- the alternating exclusion of users when working on the same objects
- the central management of users and access rights.

One server is responsible for one repository. The content of a repository can be any projects or models, and any number thereof.

No other software (such as a database management system) is required.

On-line Repository The data management provided by Innovator is referred to as an on-line repository. In this context, on-line means that every person working on a project or model is informed of the current state of the project or model at all times. This is guaranteed in that the clients are in constant contact with the repository server and immediately send modified data to the server.

This means that no "check out" or "check in" is necessary for modifying data and then making the modifications accessible to other users. This avoids significant consolidation time, which is always required for off-line operation.

In principle, any number of users can work on a project or model simultaneously. The Innovator
registered
repository is equipped with a lock concept to ensure that an object can not be changed by more than one user at the same time. Instead, it allows one user to reserve an object for modification. Other users can continue to be informed of the current state of the object, but can not actually make changes to it.

Network - Teamwork Analysis, design and implementation are usually carried out in teams. To this purpose, the Innovator architecture is designed for multi-user mode, which is based on a common data storage (by means of repository servers). Because of this, server and client programs do not have to be available on the same computer. The only requirement for working with a repository is that the client and server computers are recognized within the network and are able to establish a connection between each other. Relationships need not exist via integrated drive areas.

Neither the number of clients nor repositories participating in the network is limited. Any number of repository servers can be started in Innovator network operation, and these can be accessed by any number of users. When a user enters the interface editing mode, he or she is given the opportunity to work on any of the repositories for which servers have been started and the projects and models contained within them. In principle, regardless of the physical

location of the data within the network, this allows every Innovator user to work on any project or model. Innovator user and access rights management provides control of the access to and authorization within individual projects and models.

Platforms (independent and flexible) Innovator is available for the following operating systems, based on the individual graphical user interfaces, Windows, Presentation Manager and OSF Motif:

- Windows 95 and Windows NT
- OS/2
- UNIX (AIX, Solaris, HP-UX, SINIX)
- AlphaVMS

All features are available for all of the operating systems. The user interface is the same except for a few, marginal differences, which are based on the specific interface systems. All systems can be run within a heterogeneous network and can access the same data stock.

API, adapting to existing systems The Innovator repository allows data to be archived in a well-defined structure, which defines data types and their relationships interdependently. This repository meta-model is freely accessible and can be used to evaluate data from the repository in any manner and to insert data into the repository.

Innovator provides an interface to this purpose, which supports the Tcl script language (Tool Command Language). Tcl is public domain and is available for all of the platforms supported by Innovator.

Tcl is used within Innovator itself for ensuring the availability of generated material for example for SQL scripts for data modeling, for connecting external implementation tools and text editors, etc.

Tcl scripts can either be appended as menu commands within the framework of an Innovator shell or executed as independent programs.

2.2.2 UML Nice

Provided by	Intecs Sistemi
Used in first phase	No
Supported platforms	Unix, Windows, Java virtual machine
Repository interface	CORBA, JAVA

UmlNICE overview UmlNICE is an integrated set of tools conceived and implemented by Intecs Sistemi to provide full support for the Unified Modelling Language (UML).

The major features of UmlNICE can be summarised as follows:

- Full support for the UML notation.
- Full portability and interoperability across a wide variety of hardware and software platforms; UmlNICE is fully implemented in Pure Java thus making the toolset available on virtually any platform, including Windows and Unix.
- Scalable, component based architecture. All the major components of the toolset (Desktop, Editors, Generators, Analysers, System Model Managers (SMM) and System Models (SM)) are implemented as CORBA objects, thus making it possible to take full advantage of the benefits brought by a standard distributed object computing platform, including: transparent distribution (also over intranets and the Internet through the IIOP protocol), ease of extension and openness.
- Flexible CORBA-compliant Component Communication Infrastructure (CCI). Two implementations of the CCI are available: a lightweight, single address space version, which supports the single user version of the toolset; and a full-fledged CORBA implementation, including a commercial Object Request Broker (ORB), which supports the Enterprise version of the toolset that provides multi-user, multi-site support.

UmlNICE is packaged in three configurations:

- Entry level: implemented as a lightweight single user version of the toolset, provides a complete set of modelling functionalities.
- Professional: extends the Entry level version with generation and analysis capabilities, including code and documentation generations (see below).
- Enterprise: extend the Professional version with multi-user, multi-site functionalities. This version exploits the availability of a full-fledged CORBA Component Communication Infrastructure to provide sophisticated extensibility and integration mechanisms.

The architecture of the toolset is illustrated by Figure 4:

The Desktop The Desktop provides System Model management functions, such as creation and deletion, and permits browsing through the existing models in order to select the one to open by activating the other tools. The browsing is performed by specifying a Repository Host, which can be any computer reachable through a LAN, an Intranet or even the Internet, and navigating through the File System visible from such a Server Host. System models of frequent use can be bookmarked for quick access.

The Editors Different editors are provided to support the various diagram notations of UML: Use Case diagrams, Class diagrams, Statechart diagrams, Activity diagrams, Sequence diagrams, Collaboration diagrams, Component diagrams and Deployment diagrams. Furthermore,

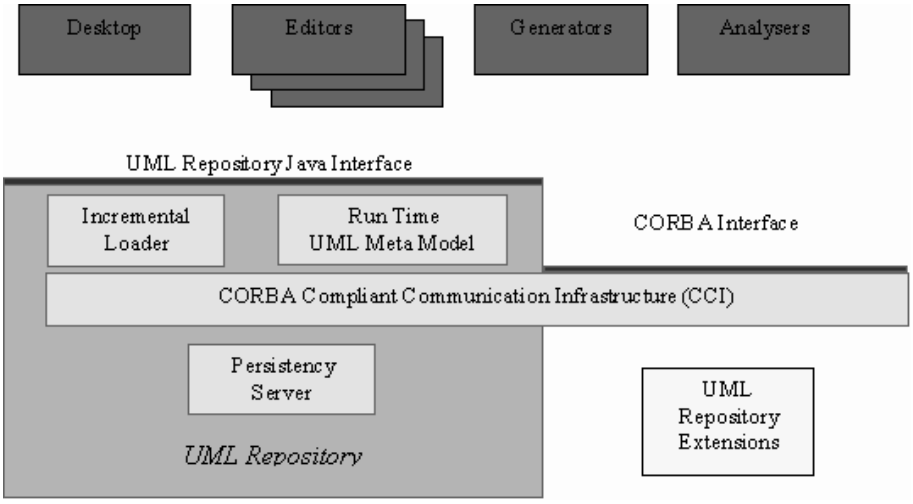

Figure 4: Analysis of an UML model

through sophisticated support for the UML extensibility features (Stereotypes, Constraints and Tagged Values) new Editors can easily be added for supporting customised UML notations.

The Editors share a common System Model Browser which supports navigation within the System Model independently from the different diagram views.

Figures 5-8 show the browser, a Use Case diagram editor, a Class Diagram editor and a Statechart diagram editor activated on the state machine associated to a class (Order).

The Generators Under the term Generators we classify a large number of tools: Code Generators, Documentation Generators, Report Generators including Off-line checkers, Textual Interchange Format Generators, etc. All these tools share a common technology: a powerful scripting language. This language provides high level, flexible and expressive primitives for navigating a System Model and for extracting information in the desired format (code, documents, reports, etc.). The availability of such a language makes it easy for the users themselves to develop new generators or to customise existing ones. In order to make this easier for expert UML users we have decided to base this language on a UML notation: the Object Constraint language (OCL), a high level powerful navigation language, and extend it with powerful generation primitives.

The Analysers The main purpose of the tools in this group is to analyse data external to the UmlNICE Repository and to import some form of representation of this data into the Repository. The external data can be source code files, Textual Interchange Format files, other tool repositories, etc. The imported representation can be a translation of the external data or a set of traceability links among UmlNICE Repository entities and external data components. Tools of this group are: Reverse Engineering tools, Model Importers, Model translators, etc.

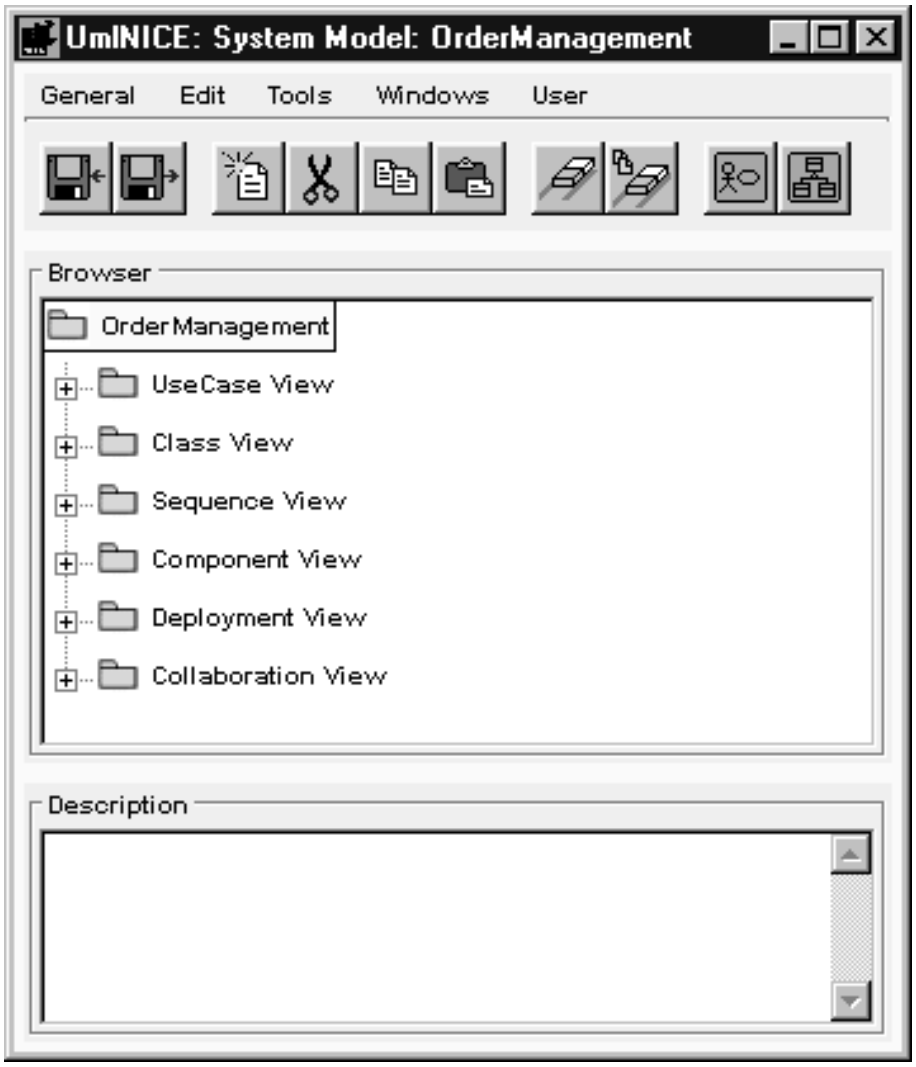

Figure 5: The Model Browser

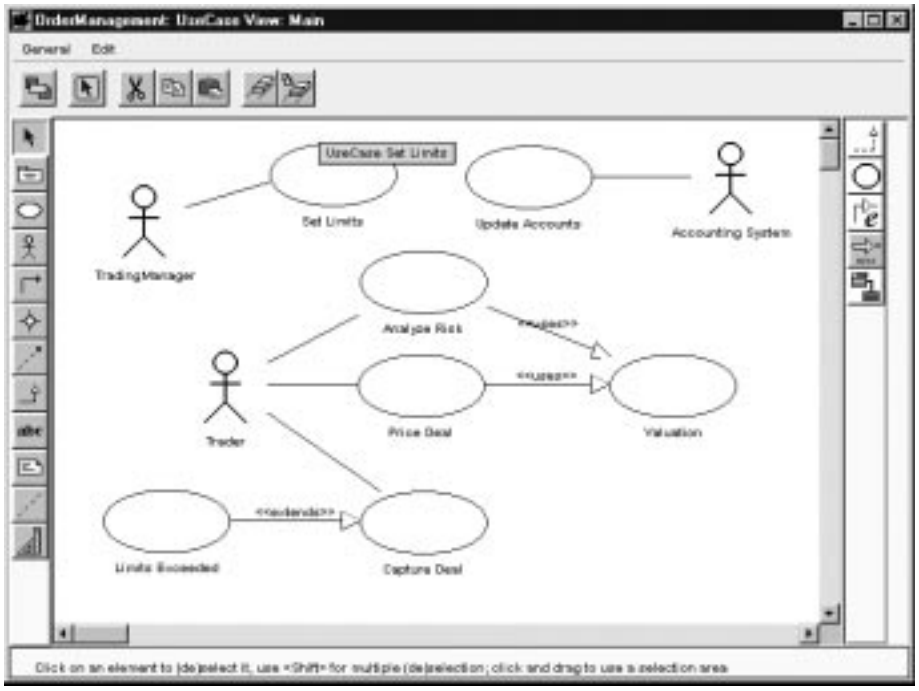

Figure 6: The Use Case Diagram Editor

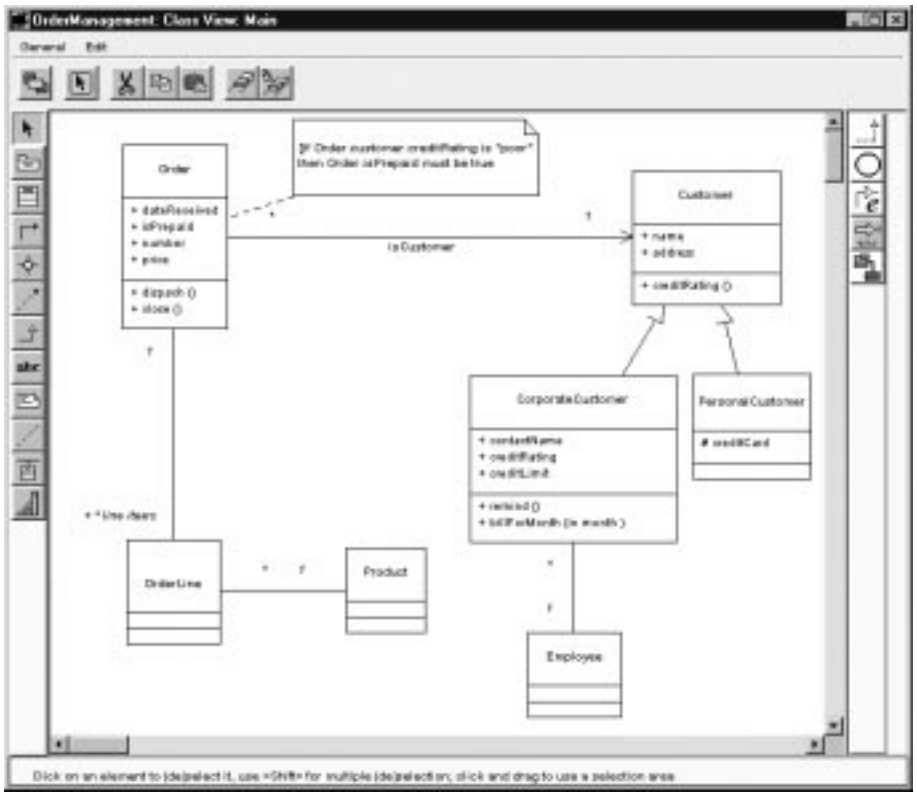

Figure 7: The Class Diagram Editor

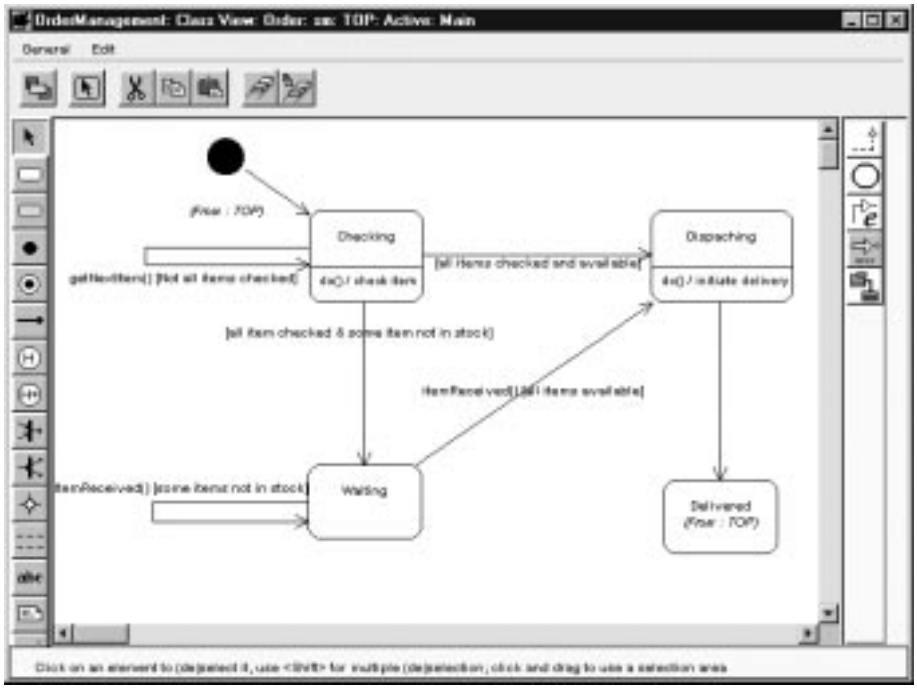

Figure 8: The Statechart Diagram Editor

The UmlNICE Repository The Repository provides classical services for the management of model data, which include:

- Management of System Models: creation, deletion, etc. These functions are performed by the System Model Managers (SMM).
- Manipulation of System Model data, namely all the functions for retrieving and modifying the internal System Model representation, including concurrent access control, access rights validation and transaction services. These functions are implemented by the System Model objects (SM).

Two Interfaces to the Repository are provided:

- a CORBA Interface, which is the IDL specification of the Repository services;
- a Java Interface which is implemented on top of the CORBA interface and extends it with additional functions which make transparent the loading of model data and their management in the run time memory.

Flexible and powerful mechanisms for extending Repository functionalities are provided: in addition to standard CORBA facilities, the possibility of capturing any communication between the tools and the Repository and of executing Pre and Post access actions is provided. These extension mechanisms are also completely available to third-party providers and to end users.

Extensions to Repository services which are planned include: System Model Dictionaries for managing libraries of reusable components, Process support engines, and Gateways to other repositories.

UmlNICE Extensibility mechanisms One of the principle objectives in the design of UmlNICE has been easy extensibility and easy of customisation. To achieve this objective, a number of mechanisms are provided to the user for supporting the customisation of the tool and the development of new functionalities that can be easily integrated with UmlNICE.

The major mechanisms are the following:

- Full support to the UML extensibility mechanisms, as stereotypes,tagged values and constraints. UmlNICE allows the user to easily define new stereotypes and to associate them to any model element. Model elements can be annotated with tagged values and with constraints expressed in the OCL.
- The OCL interpreter integrated with UmlNICE allows to execute the constraint expressions. The OCL language has been extended adding the possibility to invoke Java methods from within an expression. This allows to easily generate reports or in general to extract information from the Repository while an expression is evaluated.
- Activation of external tools. Through the User menu available both from the Desktop and the Browser, external tools can be activated. The mechanism provided allow both the activation of an external process and the invocation of a method of a Java class.
- Availability of API for accessing the major tool services. In particular the repository services are made available both through a Java API [5] and a CORBA IDL API.
- Availability of a Generation Framework which implements a Visitor pattern on the Repository. This Framework allow to navigate the repository in a transparent way and to code just the semantics actions that have to be executed when a node is visited.

2.3 Verification tools

2.3.1 Spin

Provided by	Bell Labs
Used in first phase	Yes
Supported platforms	UNIX, Windows95 or WindowsNT

General Description Spin is a widely distributed software package that supports the formal verification of distributed systems. The software was developed at Bell Labs in the formal methods and verification group. Some of the features that set this tool apart from related verification systems are:

[5] The detailed documentation of the Java API can be obtained from the tool provider

- Spin targets efficient software verification, not hardware verification. Spin has been used to trace logical design errors in distributed systems design, such as operating systems, data communications protocols, switching systems, concurrent algorithms, railway signaling protocols, etc. The tool checks the logical consistency of a specification. It reports on deadlocks, unspecified receptions, flags incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes.

- Spin works on-the-fly, which means that it avoids the need to construct of a global state graph, or a Kripke structure, as a prerequisite for the verification of any system properties.

- Spin can be used as a full LTL model checking system, supporting all correctness requirements expressible in linear time temporal logic, but it can also be used as an efficient on-the-fly verifier for more basic safety and liveness properties. Many of the latter properties can be expressed, and verified, without the use of LTL.

- Correctness properties can be specified as system or process invariants (using assertions), or as general linear temporal logic requirements (LTL), either directly in the syntax of next-time free LTL, or indirectly as Büchi Automata (called never claims).

- Spin supports dynamically growing and shrinking numbers of processes, using a rubber state vector technique.

- Spin supports both rendezvous and buffered message passing, and communication through shared memory. Mixed systems, using both synchronous and asynchronous communications, are also supported. Message channel identifiers for both rendezvous and buffered channels, can be passed from one process to another in messages.

- Spin supports random, interactive and guided simulation, and both exhaustive and partial proof techniques. The tool is meant to scale smoothly with problem size, and is specifically designed to handle even very large problem sizes.

- To optimize the verification runs, the tool exploits efficient partial order reduction techniques, and (optionally) BDD-like storage techniques.

To verify a design, a formal model is built using PROMELA, Spin's input language. PROMELA (a Process Meta Language) is a non-deterministic language, loosely based on Dijkstra's guarded command language notation and Hoare's language CSP, extended with powerful new constructs.

Spin can be used in three basic modes:

1. as a simulator, allowing for rapid prototyping with a random, guided, or interactive simulations

2. as an exhaustive state space analyzer, capable of rigorously proving the validity of user specified correctness requirements (using partial order reduction theory to optimize the search)

3. as a bit-state space analyzer that can validate even very large protocol systems with maximal coverage of the state space (a proof approximation technique).

The Spin software is written in ANSI standard C, and is portable across all versions of the UNIX operating system. It can also be compiled to run on any standard PC running a Windows95 or WindowsNT operating system.

2.3.2 Panda

Provided by	University of Erlangen-Nürnberg
Used in first phase	Yes
Supported platforms	UNIX

PANDA (Petri net ANalysis and Design Assistant) is a software tool for the analysis of timed Petri nets which is being developed at the Computer Science Department of the University of Erlangen- Nürnberg.

Some of the new capabilities of PANDA are:

- Development of faster algorithms. PANDA uses specially developed methods for the solution of the underlying mathematical problems in order to decrease the user's turnaround time.

- Inclusion of optimisation methods. Since modelling tools such as Petri nets are often used at the design phase, the ability to automatically perform optimisation according to selected target functions and boundary conditions will be an attractive feature of the software package.

- General distribution functions. The standard type of timed Petri nets commonly found in software tools (so-called GSPNs) is characterised by allowing only exponentially distributed firing times. In practice, however, one frequently requires more general distributions. Thus one objective of PANDA is to allow the use of these functions. Since this prevents the use of numerical solution methods, PANDA also provides discrete simulation techniques.

- Parallel computing. The mathematical analysis of Petri nets frequently requires the solution of extremely large systems of equations. The ability to use modern parallel architectures will give access to the substantially larger memories and computing performance of these machines.

- Fault models. PANDA includes additional fault-modelling tools for designing success diagrams and fault trees and converting them automatically into Petri nets.

  These graphs provide much easier handling of complex systems while retaining all the advantages of Petri nets.

Components	Requirements
CASE tools	All UML features which are supported and used by the HIDE framework Extensibility features for the end-user: internal script language Requirements of the script language: Reading interface for the CASE tool's UML repository; Writing interface for the CASE tool's UML repository; Input-output features; Optional: supporting database connections; Optional: development of graphical interface
Databases	Relational database PL/SQL as internal DML language Storage of PL/SQL packages Various platforms Client-server architecture Reachability from clients from differnt platforms
Verification tools	Known input file syntax Known internal representation Batch mode execution

Table 2: Requirements for the HIDE components

PANDA offers the user a graphical interface for the input and editing of the Petri nets. The user can define his or her own evaluation criteria based on inherent properties of the net. Both transient and steady-state analysis are available, which can be computed via numerical or simulative algorithms. The results of the analysis are presented in textual and graphical form. Although it is still under development, PANDA is already in research use at Erlangen University.

2.4 Fault-tolerant components library

Not implemented in the Phase 1. However, the database system based HIDE repository provides an easy interface for implementation of the fault-tolerant components library inside the database

3 Requirements on external tools

As the HIDE framework uses many off-the-shelf components and does not have the possibility to alter them for its need, it is obvious that some requirements must be formulated for the integrated external tools. These requirements are summarized in Table 2.