AMEISE – A Media Education Initiative for Software Engineering
Concepts, the Environment and Initial Experiences

R. T. Mittermeir*), E. Hochmüller**), A. Bollin*), S. Jäger*), M. Nusser**)

*) Universität Klagenfurt, **) Fachhochschule Technikum Kärnten

Key words: *Tools for interactive learning and teaching, computer based learning (CBL), software project management*

Abstract:
An experience-dominated subject like software project management cannot be learned by merely attending lectures. Additional labs, however, even with only modest real-life projects, call for substantial effort to be spent by the instructors as well as by the partaking students. The AMEISE[1] approach provides an efficient and effective education alternative. Based on Stuttgart University's SESAM, it allows for repeatedly experiencing the complexity of software project management within a simulation environment.

In an AMEISE simulation, students assume the role of a technical project manager. They can hire and fire personnel, structure the project, and allocate tasks. Students are challenged to manage a project according to a particular model of the problem structure, selected by the instructor. It is up to the instructor to select the number of trials (simulation runs) to solve given tasks within specified constraints. Students can learn from previous simulation runs, change their strategies and measure their own success using the AMEISE self-assessment feature.

This paper describes the underlying AMEISE concepts (including model structure, constraints, and role concept), the architecture and components of the AMEISE environment, and assessment possibilities. Some initial experiences gained from software engineering courses using the AMEISE environment will be discussed as well.

1. Motivation

Educational institutions are increasingly challenged to present not only generally valid foundations of the scientific discipline they are lecturing in. Students need to be prepared for the reality of industrial life by showing them real practice. But what is this practice the labor market is demanding from young graduates? Is performing ten times the same routine activity, an activity graduates have to perform in their job an uncounted number of times, practice? Is practice to have students work in a real project that is in some extreme situation? Is it to have them to work in some five projects in a guided manner, so that they don't only experience a lot, but also consciously observe what they have been experiencing?

[1] AMEISE is a joint project of Universität Klagenfurt, Fachhochschule Technikum Kärnten, Universität Linz und Universität Stuttgart. It is funded by the Austrian Science Ministry, BM:BWK, under grant NML 1/77.

We all know the answer to these questions. But to live up to it is hard if not impossible. Where to get the range of situations from? Which company allows studying its failed projects? Who is ready to entrust the management of a difficult project to persons who have not even completed their education? How to determine ahead of time, what interesting aspects will pop up in a project several months from now? And finally: how to integrate students in a project which has a history that extends the duration of a course or even of a full curriculum? Constraints educational institutions have to live with limit the richness in which real practice can be demonstrated.

Nevertheless, although the situation is difficult, it is surmountable with modern educational technology. At least, if one accepts that, working in real projects can be seen as a multi-facetted activity. Some of the facets are strictly organizational and psychological, some of the facets are technical, structural, or have just to do with the complexity of multi-criteria decision making under incomplete information. Some of the latter aspects can be captured by modeling the application domain and presenting it to the student in a complex simulated scenario. The SESAM approach [1] follows this principle. The AMEISE-system builds on SESAM and extends it to new educational situations.

The rest of the paper is structured as follows. First, the educational principles behind SESAM and AMEISE are presented. Based on a discussion of technical challenges, chapter 3 introduces the AMEISE architecture and its core components in charge for the general infrastructure. The next chapter is devoted to the so-called explanation components which represent a substantial conceptual value added by AMEISE. After the discussion of some experiences already gained from AMEISE simulations so far, some conclusions are drawn.

2. SESAM and AMEISE

SESAM, Software Engineering Simulation by Animated Models, presents students with a kind of software engineering adventure game. Students get a project of a given size (estimated adjusted function points) assigned. They have to assume the role of a project manager, aiming to complete the project within given time and budget by a team of simulated, virtual software engineers. Students can hire personnel form a pool of persons with different qualifications and different expected salary. The product (code and documentation) has to meet certain quality levels (completeness, faults per LOC or errors per page in the documentation). The characterization as adventure game is justified in so far, as SESAM is free from prescriptive features. Students are in full control concerning the tasks they are giving to their virtual employees. However, this requires planning, since otherwise, costly idle time or wasted effort due to error correction, lack of focused work or other less than perfect situations may result. On the other hand, there is no way to pre-compute an optimal trail through the project. Taking decisions always means arbitrating between competing objectives.

The developers of AMEISE, A Media Education Initiative for Software Engineering, appreciated the core ideas behind SESAM. However, they aimed at extending the spectrum of educational situations where this project management adventure game can be played. E.g., in addition to acting as “lone hunters,” students should have a chance to enter competitive team situations by allowing them to compare their performance relative to the performance of peers. Further, students should have a chance to explore different trails without re-walking the full project trail. Last but not least, we wanted to relieve the instructors form some of the less complex interpretive tasks. This might get them a chance to better focus on complex interactions between effects.

Some of the AMEISE objectives are:

- ***O1:*** *You are allowed to fail once*. This is already a SESAM principle. The complexity of the first project usually overwhelms students. Hence, they may redo it after having studied, why they missed some or most criteria in their first trial. In AMEISE, the time between these runs is shortened, since students get most “single-mistake”-reports directly from the system.
- ***O2:*** *You may experiment*. In real projects, decisions taken are irrevocable. Only some can be compensated by reverse commands. In an AMEISE-project, the instructor may define milestones where students may restart the project. Thus, they can study the effects of different managerial decisions quite easily.
- ***O3:*** *Make education a competitive game*. There is no ideal project. But there are better or worse solutions and it is not only worthwhile but also motivating to compare ones performance with those of peers.
- ***O4:*** *Sustainability*. Developing educational software such as SESAM/AMEISE is expensive. Hence, one wants to use it in various educational situations. To allow for this, a suite of different simulation models could be built. To validate them in reality will be difficult. AMEISE rather assumed the approach to support students by a set of parametric agents. “Friendly peers” will look over the shoulder of the young project manager and become active when apparently very severe mistakes are in the offing. “Consultants”, on the other hand, have to be invoked by the student, if the management-trainee realizes that a situation has developed, where some additional help is needed.
- ***O5:*** *Interpretation.* SESAM monitored the student’s activity. By rerunning the simulation in an instructor’s mode, analyses could be obtained. In AMEISE, no rerun is needed. Monitoring is done in a database that can be queried right away. The results of such queries are directly fed into a graphical and text-based evaluation tool.
- ***O6:*** *User interface*. SESAM has a text based, pseudo-natural language user interface. We decided to replace this by an interface the current PC-generation of students might be more used to.

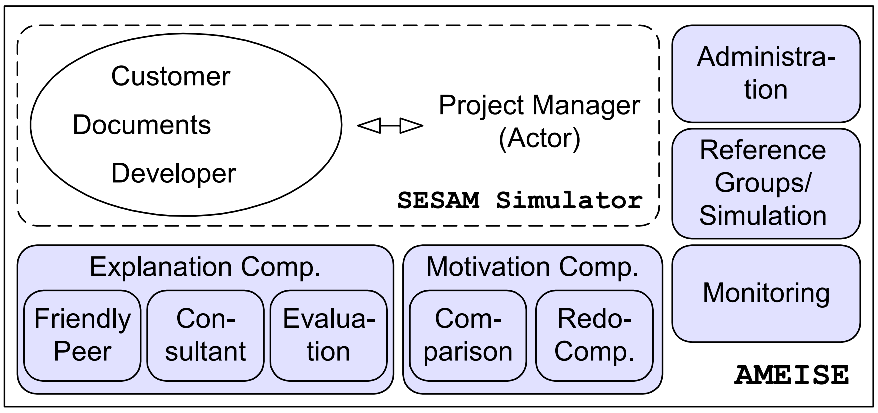


Fig. 1: AMEISE extensions to SESAM. The SESAM core is embedded in the AMEISE environment. AMEISE adds administrative components as well as evaluation and explanation components to the simulator.

Fig. 1 shows the difference between AMEISE and SESAM in a nutshell. The SESAM system is still used to provide the basis for the simulation. But AMEISE makes also use of the SESAM states and simulation data (“Reference Groups/Simulation”). It monitors the actors’ progress, collects performance data, and stores this data in a database (“Monitoring”).

Furthermore, it provides components for the explanation of simulation results ("Friendly Peer", "Consultant", and "Evaluation" for inspecting simulation results) as well as components supporting different educational principles ("Comparison" for comparing project states, "Redo" for a roll-back of management-decisions). The following chapters introduce the architecture of the AMEISE system that deals with these requirements.

3. Architecture

The objectives mentioned motivated us to change the overall architecture of the system. While SESAM is a single user system, AMEISE is based on a distributed architecture. For various organizational reasons, the simulation engine of SESAM has been wrapped. A load balancing manager has been introduced as middleware linking various clients to possibly different wrapped SESAM servers. The common state of the diverse simulation runs is kept in a central database. This database and the load balancing manager can be considered as the backbone of the AMEISE architecture. The specific design considerations that led to this architecture are explained in the following sections.

3.1. Challenges

Chapter 2 introduced several AMEISE objectives which cannot be fully met by SESAM. From these objectives, requirements at a more technical level can be deduced:

- *Allow rollback of decisions and comparison.* Objectives O1 and O2 imply that in contrast to SESAM, old simulation states and related key-data (e.g. the number of errors in a document or the number of errors found in a review meeting) have to be stored in order to enable the actor to undo decisions and compare the implications of different decisions. In AMEISE, all simulation states (and related data) are stored in a *database* and can be mapped back to the SESAM simulator (thus restoring a former state) or queried by the AMEISE system for further evaluation and comparison.
- *Provide a multi-user environment.* SESAM per se is a single user environment. In order to create competitive situations (objective O3), the system has to have access to results or key-data of other simulations. The *database* is used as an interface between SESAM simulations, performance data, and simulation results.
- *Support the user.* SESAM has a simple text-based user interface which requires a UNIX environment. In order to ease the operation of the simulator and provide full-graphical feedback, a Java based graphical interface has been implemented. The advantage of this approach is that the interface is available on different platforms. This enables the system to run on a Windows-based environment. Furthermore, advisory components (running as Java-agents collecting and evaluating key-data) have been introduced. In order to have the key-data available at all times, even after a restart of the system, these components are storing their "knowledge" also in the database.
- *Support the tutor.* The same holds for objective O5: the support for the tutor of the course. Analyzing the simulation by hand is a time-consuming task. AMEISE supports the tutor by storing all relevant data in the database during the simulation. This means that a final evaluation can be done by simply executing pre-defined queries. The database is much more than only storage for states; it really provides the basis for all of the explanation components like the consultant or friendly peer (cf. chapter 4). The abilities of the database and the AMEISE database scheme will be explained in more detail in section 3.3.

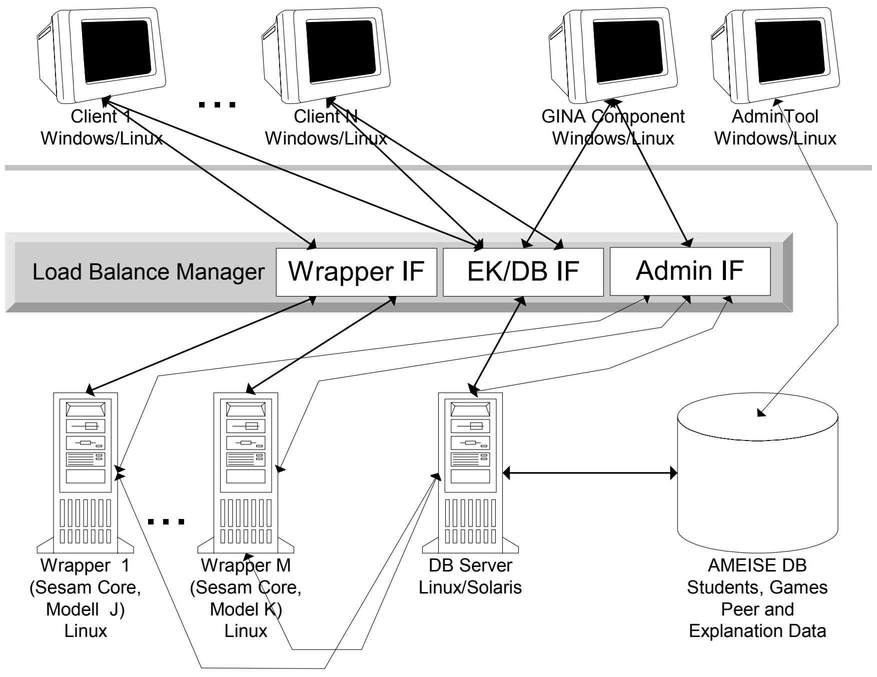


Fig. 2: The AMEISE system consists of a set of Wrappers for the SESAM engines, a Load-Balance Manager for scheduling Client requests and a series of AMEISE Clients. AdminTool and GINA are administrative components for configuring and monitoring the system.

The SESAM system itself has been implemented in Ada 95 [2] and TCL/TK [3]. This means that every system running the simulator requires suitable Ada and TCL/TK environments. The technical requirements strongly suggest the use of a database in the background and, for the user, a client interface that is implemented in Java, so that the interface can easily be used on different platforms without modifications. On the simulation level Ada 95 and TCL/TK complicate the use and distribution of the system. In fact, there are two other, implicit requirements influencing the architecture of the AMEISE system:

- *Allow AMEISE to scale-up to larger courses and to different platforms.* Simulating a SESAM model requires a huge amount of resources, resources a simple desktop computer is not able to provide. As it is not feasible to provide a separate server for each user or each model of AMEISE, the system is designed as a *Client-Server system*, where requests of different clients (AMEISE users) are scheduled by a so-called *load-balancing* component and routed to a free or less occupied simulation engine. Adding new models or scaling up for more users is simplified, as simulation engines can be added to or removed from the system at runtime. Clients are communicating with the simulation server in the form of Java Request Objects across the Intra- or Internet and at installation time the user does not have to deal with suitable Ada or TCL/TK environments.
- *Stay compatible to the SESAM environment.* As already mentioned (objective O4), developing simulations (or models, as a simulation model describes a specific kind of simulation) is expensive. Therefore, one major objective was to make use of existing and future SESAM simulations and developments. We decided not to alter/adopt the basic simulation engine, but to integrate the SESAM simulator as is into the AMEISE system by *wrapping* the SESAM system and by providing a simple Java-based interface. This means that we still have to stick on Ada, but the simulation core can be

compiled into a shared library with only minor adaptations on the compiler-switch level. This library can be pre-compiled for different platforms and simplifies the installation of the whole system.

Based on the requirements mentioned the AMEISE system (see Fig. 2) consists of three major components a portable *AMEISE client*, a *wrapper* encapsulating the SESAM simulator and a *load-balance manager* (LBM for short) for scheduling client requests. Further, it comprises several tools for administrating the system.

3.2. Components

The AMEISE system is designed as a client-server system with two peculiarities. Firstly, multi-user functionality of the simulator (originally designed for single-user operations) is achieved by the wrapper. It maps different simulation states from and into the simulation engine. Secondly, the client is stateless in order to cope with Java malfunctions or problems with the Intra-/Internet connection.

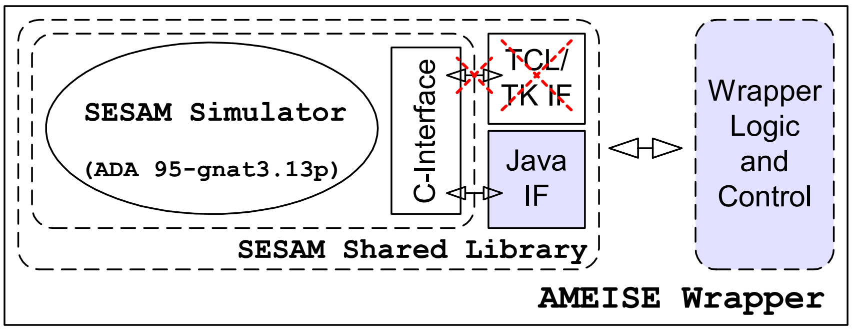


Fig. 3: The wrapper encapsulates the SESAM kernel implemented in Ada 95. By replacing the TCL/TK interface by a Java JNI interface, the wrapper is able to access all simulation functions. The interface is compiled into a shared library, so no local ADA or TCL/TK installation is necessary for running AMEISE.

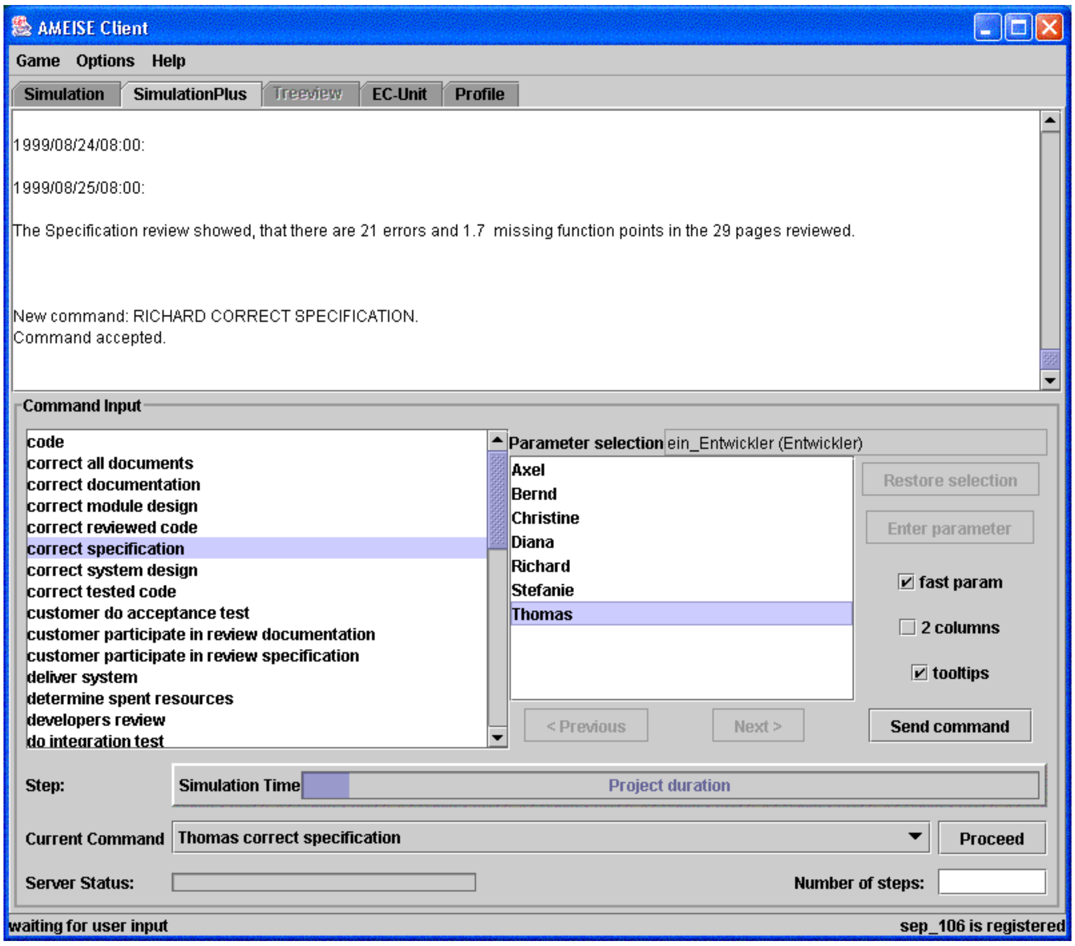
The central AMEISE-component (Fig. 3) is called *wrapper*, and it is responsible for three major tasks:

- it encapsulates the SESAM simulation engine (which is bound to a specific simulation model) and provides suitable interfaces to it;
- it receives client requests (via the LBM), retrieves the old state from the database, sends the request to the simulation engine, and inserts the new states into the database;
- it collects relevant state data (model data) and sends this data and the response of the simulation engine back to the client.

The wrapper is fully implemented in Java and loads the SESAM simulator via a shared library. It is initialized by one particular simulation model. Therefore, one wrapper can handle only one kind of simulation at a time. But as states of different users can be loaded into the simulator, the wrapper can be seen as a component providing over-all multi-user capabilities (per model) to the SESAM system.

The second component is called *LBM* (*load-balance manager*) and its task is to manage a set of wrappers (and hence also models). It passes client requests to a free (suitable) wrapper, but also processes the response of the wrapper. The response of the simulation engine is passed back to the client. Thus, the collected state-data is analyzed and stored in the database. Last but not least, the LBM is also the mediator between the client and the database. Some components of the client (e.g. the component providing comparison data) need access to the database and the LBM serves as a tunnel. This enables the system to work through a firewall, as only the LBM-port has to be opened and the database does not have to be accessible from all over the world.

The third major component is the *AMEISE client*. It is implemented in Java using Java Swing components for the user interface. The main task of this component is to provide a graphical user interface (GUI) such that the student is able to interact with the AMEISE system. The user has the possibility to choose between two different user interfaces, the pure "Simulation" interface which corresponds to the look and feel of the original TCL/TK SESAM interface and the "SimulationPlus" interface which is represented in Fig. 4. The latter provides a model-specific list of all possible commands which can be executed by a student in the role of a project manager. Each command can be constructed by selecting the appropriate command from the "Command Input" list and choosing the desired parameters from a similar list which is automatically derived from the context of the command previously selected. The commands in Fig. 4 are derived from the so-called *Quality Assurance* model (*QA* model) which was already developed by the SESAM team. It implements a phase-driven simulation setting with appropriate quality assurance activities like review, test, as well as correction of life cycle documents (e.g. specification, system design, module design, or code). A detailed description of the QA model is given in [4]. However, the client is able to parse any syntactically correct model information (contained in a so-called *dictionary*) resulting in automatically derived model-specific command lists.



1999/08/24/08:00:

1999/08/25/08:00:

The Specification review showed, that there are 21 errors and 1.7 missing function points in the 29 pages reviewed.

New command: RICHARD CORRECT SPECIFICATION.
Command accepted.

Parameter selection: ein_Entwickler (Entwickler)

[✓] fast param
[ ] 2 columns
[✓] tooltips

Current Command: Thomas correct specification

waiting for user input
sep_106 is registered

Fig.4: *SimulationPlus* client user interface

Other important tasks of the client are:

- It communicates with the wrapper. As mentioned before, the AMEISE system has a client-server architecture. For this reason it is necessary, that the client and the server permanently communicate with each other. This communication deals with user information and simulation data.
- It offers various user interfaces. Some interfaces allow the student to run a simulation ("Simulation", "SimulationPlus"), others support student-activities by providing user data and statistics ("Profile") and information generated by parametric agents (explanation components – "EC-Unit").
- It cares for user identification which guarantees that the user is the only one that is able to run his own simulation.
- It allows the user to start or stop simulation runs whenever wanted. The user has the chance to run different simulations and to start or stop them whenever considered appropriate. Thus, the client enables the user to choose the simulations he wants to start or to continue.

In fact, there are also two further components for the administration of the system:

- The so-called *AdminTool* component enables the tutor to manage users and courses, to insert new kinds of models to the system, but also to manage (and extend) the abilities of the explanation components.
- The *GINA* component is a small tool for monitoring the system, especially at the system request level (e.g. response queues) and provides some statistical data.

3.3. Database

The AMEISE database serves as a central archive to be used by other AMEISE components. In Fig. 5, the conceptual data model is represented as an OMT object diagram (using cardinality notation for association multiplicity). It can broadly be divided into 5 groups of classes which are semantically related to each other. The different colors used in Fig. 5 (also electronically available from http://ameise.uni-klu.ac.at/publikationen/dbschema_v1.0.7a.pdf) should help in identifying the classes of each of these groups or sections:

- *User Information (blue):* AMEISE allows for the administration of users who can use the system based on a particular role model. At the moment, we distinguish between administrator (full access rights to the database), model designer (defines different kinds of simulation situations), instructor (leads a simulation course), and player (student who can run simulation(s)).
- *Course Information (green):* This group contains data about courses, their instructors and students (in the role of assigned managers). As AMEISE simulations can be carried out via the internet by partner institutions, some data characterizing these organizations using the AMEISE server can be stored, too.
- *Model Information (red):* Different kinds of simulation situations can be defined as so-called "models". Valid simulation commands are stated in the so-called "dictionary" which is used by the client GUI (SimulatorPlus) to derive the command list. The data includes also the text describing the simulation problem to be solved by the student.
- *Explanation Information (orange):* This section contains model-specific parameters which are used by various explanation components (as described in chapter 4). These components are regarded as so-called different "kinds of aid". For each model, they can be parameterized by the concept of "specific aid" with related rules and queries. The latter will be evaluated during runtime of the explanation components

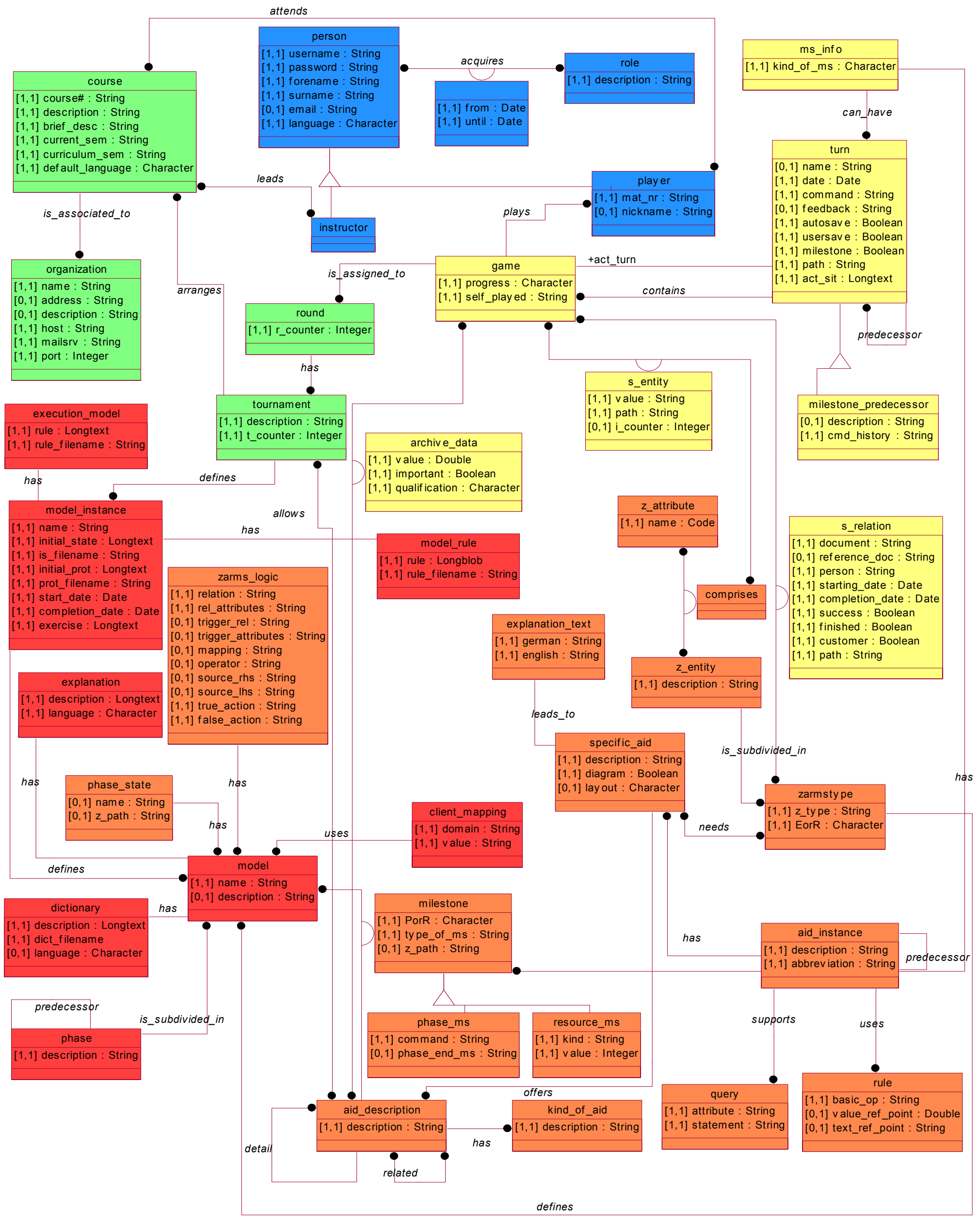


Fig. 5: AMEISE Object schema

called by the user (instructor or student). Thus, the designer of a new model only has to provide the explanation parameters leading to explanation components which will be ready to use without any additional coding effort. Therefore, the explanation components can be characterized as being parametric agents.

- *Simulation Information (yellow):* Once an instructor has defined the setting for a simulation, a so-called “run” or “game” is initialized. When a student proceeds with his commands during a simulation, the student’s actions and related information is stored in the database at runtime. This information will be used by the different explanation components (cf. section 4) to allow for situation-dependent feedback.

The AdminTool component serves as the common user interface for the administration (manipulation and querying) of information related to the first four sections. It distinguishes between users with different access rights depending on their roles (administrator, model designer, and instructor). The data concerning the last section is generated and inserted during runtime of AMEISE simulations restricting the respective functionality of the AdminTool to querying and some kind of garbage collection activities, only.

4. Explanation Components

The following components are aids for the student and instructor. They represent the heart of the AMEISE system. The consultant and the friendly peer [5] can be seen as parametric agents which support the students while managing a project, whereas the evaluation component [6] gives the students the possibility to analyse their management performance. The consultant and the evaluation can be found on the clients "EC Unit" user interface, whereas the friendly peer acts like an agent and interacts with the student via the "Simulation"/"SimulationPlus" interfaces. In the sequel, these explanation components will be explained on basis of the QA model [4] as underlying simulation model. However, they are designed and implemented as generic components, so that they can be used for other models, too.

4.1. Consultant

The consultant can be regarded as a tutor or advisor. The student can ask this component about specific problems. In answering, it considers the actual situation of the project. The student-manager takes the initiative by choosing a specific question. These questions are available in a well-structured question pool. Of course, only questions that refer to project phases that were already entered by the student can be answered. In case the student asks a premature question, the consultant realizes the situation and replies that there is no valid answer possible at this moment. The knowledge necessary for answering those questions can be derived from the database, where every turn is stored.

Example: Assume, student Bob finishes the specification and just starts with the design, but asks already something about the implementation. As the implementation has not even been started the consultant replies that this question cannot be answered yet.

4.2. Friendly Peer

The friendly peer can be seen as a trouper who has a great amount of experience and is an expert in this domain. This component is keeping an eye on the student-manager and hints at severe mistakes the student is about to make.

The friendly peer runs in the background. Therefore, it is completely transparent for the student. When the friendly peer detects a mistake or a situation that will lead to a substantially better solution, the agent contacts the student. In order to notice error-prone situations, the friendly peer has to watch permanently every simulation situation during the whole simulation run. On every simulation step the actual situation is compared with a locally nearly ideal situation that is stored in the database. Locally, in this context highlights the fact that the friendly peer has no overall optimal situation at disposal. It oversees a given number of most recent managerial actions and situations. If, within this loophole, a deviation between the actual situation and an evidently better situation or strategy is detected, it warns the student.

To observe the actual situation the friendly peer attempts to derive the necessary information from the database. But as some local information is not relevant globally or not easily obtainable form the database, additional information has to be built up in local memory. With every simulation turn, the friendly peer has to check whether the local memory has to be re-initialized or updated.

Example: Observe the number of days a hired developer is not working. To realize this situation, the number of days a developer has no task assigned must be memorized. When this number exceeds a specific limit, the student's attention is drawn to this situation.

4.3. Evaluation

In order that students reach a high level of competence, the instructor has to closely look at the project's progress. It is important that the most relevant information is filtered out from monitoring data and discussed with the student(s). This helps the management-trainees understand problematic decisions during the project. The instructor can show how each decision influenced the whole project. The more information an instructor extracts from the project's monitoring-data, the better are the students supported.

However, the evaluation component will not fully replace the instructor. It just reduces the interpretative effort dramatically. The component analyzes the project's whole progress with respect to a set of assessment criteria. Not only elements of the project's goal vector, like project costs, code quality, or project duration are part of the assessment criteria. One considers among others also the completeness and correctness of documents of several project phases.

After having delivered the developed software project, the student can analyze the project's progress her- or himself. Different criteria can be chosen from a pool of assessment criteria. The selected criterion will be evaluated by the evaluation component and the result will be shown via the client subsystem.

For each assessment criterion specific attributes of the actual project status have to be analyzed. Some assessment criteria for a model focussing on early consideration of quality are:

Assessment Criterion	Question
project cost	Did the project cost remain within the limit of € 450.000?
project period	Did the project period stay within the limit of 270 days?
AFP for the code	Did the student-manager succeed in implementing at least 95 % of the required adjusted function points?
errors remaining in the code	Does the final code contain less than 12 errors per KLOC as requested by the customer's acceptance criteria?

Table 1: Some assessment criteria for the evaluation of project results

The construction of the questions shown in Table 1 exhibits a common pattern. There are always two important elements: the attribute which has to be analysed and the respective reference point (numeric or string value). Relating these by a comparison operator yields either YES or NO as intermediate result. According to this answer different evaluation texts are needed.

For this reason every assessment criterion is represented as a set of so-called *specific aids*. Each specific aid handles one of these cases. E.g., the question “Did the project cost remain within the limit of € 450.000?” can be answered with yes or no. Therefore, this assessment criterion is represented in AMEISE by two different specific aids.

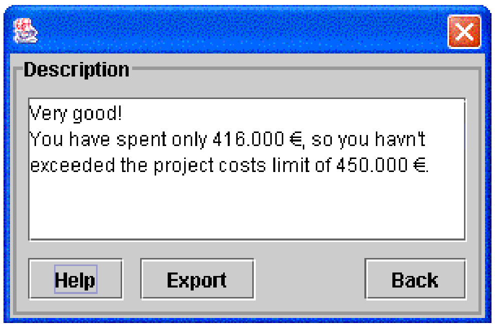
1) Total cost is greater than € 450.000.
2) Total cost is less or equal than € 450.000.

Chaining of the attribute-operator-reference_point triplets are needed to allow for more complex situations, such as “AFP_for_specification > 199 AND errors_in_specification < 30” which should yield “Your specification is complete and quite correct!” or an example dealing with task assignment to developers such as “author_specification == “Diana” AND qualified_for_writing_specifications == “Diana” which should yield “Diana was the right choice for writing the specification.”

All project attributes relevant for the evaluation of the project are stored in the database. They can be obtained by an SQL-statement which gets the value of an attribute for a given simulation run from the database.

Evaluation of an assessment criterion is carried out by a rule interpreter. Its task is to get the relevant information from the database and to formulate an automatically generated SQL-query which finally yields the explanations to be given to the student.

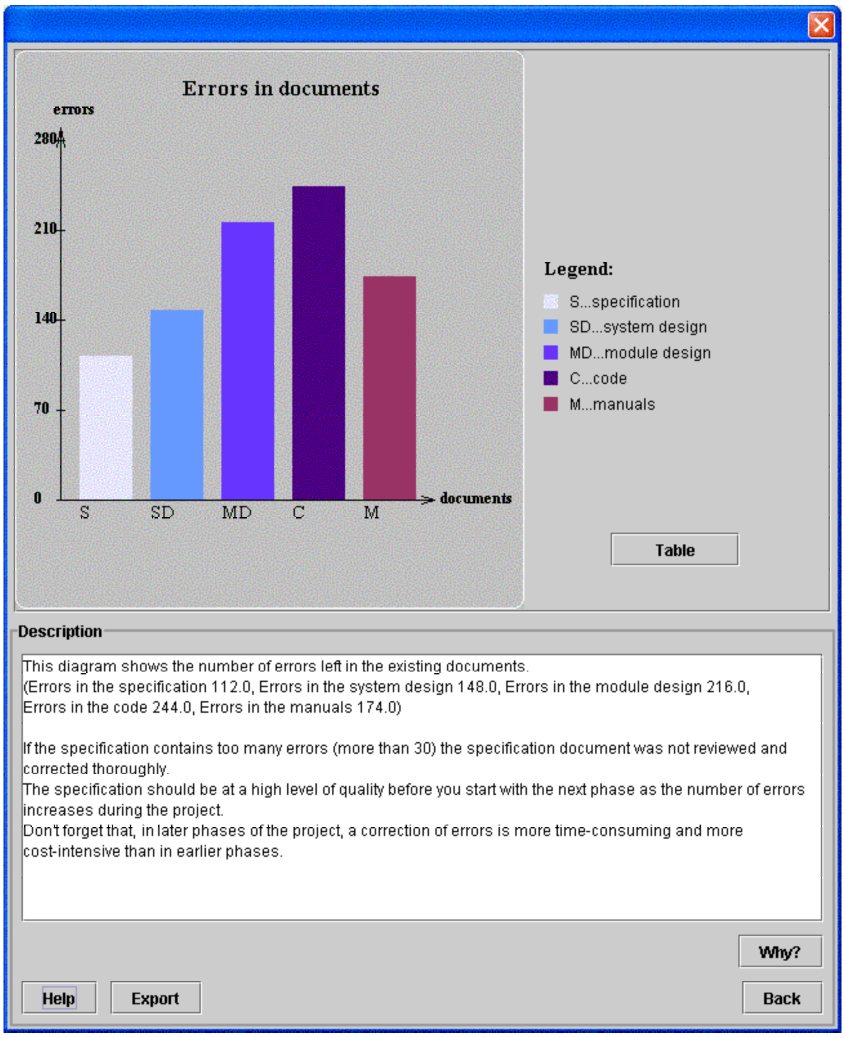
E.g., for the project cost criterion, the analysis starts with the examination of one specific aid which is related to the attribute “cost”. The rule interpreter will execute an SQL-statement to load the actual value of the attribute “cost” from the project monitoring part of the database. Assuming the actual attribute value is 416.000, it combines this value, the operator (assume “>”) and the reference point (allocated budget of € 450.000) to a new predicate. This predicate looks like: 416.000 > 450.000. Its evaluation yields FALSE. Hence, the rule interpreter aborts the evaluation of this alternative and tries the next specific aid. In this case, this will be composed to the predicate 416.000 <= 450.000 which evaluates to TRUE. In this simple case, there are no more rules in the rule chain. Therefore, the examination of this specific aid terminates successfully and the message associated to this (single element) chain of rules will be passed to the client for display (Fig.6).



Very good!
You have spent only 416.000 €, so you havn't exceeded the project costs limit of 450.000 €.

Fig.6: Explanation text of the assessment criterion „project costs”

To cover all possible situations that might emerge in an AMEISE simulation, the construction of the specific aids has to ascertain that every possible value of the attributes in the rule is covered by a simple rule or a chain of rules leading to a meaningful message. In addition to textual messages, most assessment criteria provide also graphical feedback. Fig. 7 shows this combination of evaluation text and diagram with the assessment criterion “Errors in documents”. The student can see at a glance that the quality of the documents decreases, because the number of errors in documents increases during the software development. Inspecting other assessment criteria will help to find the causes for this phenomenon.



This diagram shows the number of errors left in the existing documents.
(Errors in the specification 112.0, Errors in the system design 148.0, Errors in the module design 216.0, Errors in the code 244.0, Errors in the manuals 174.0)

If the specification contains too many errors (more than 30) the specification document was not reviewed and corrected thoroughly.
The specification should be at a high level of quality before you start with the next phase as the number of errors increases during the project.
Don't forget that, in later phases of the project, a correction of errors is more time-consuming and more cost-intensive than in earlier phases.

Fig. 7: Evaluation of an assessment criterion with a diagram

In future versions of the evaluation component it should be possible that the analysis can also show influences between different assessment criteria. This will allow a dialog between student and system, explaining which particular management decisions led to a specific effect.

5. Process and Experiences

AMEISE is an ongoing project that used as its core the software developed in another project (SESAM). This allowed us to have students work with the system and thus gain experience while developing further extensions. To report the details of the project-management results achieved by the students would be beyond the scope of this paper. Hence, this section rather explains how using SESAM/AMEISE shaped the overall progress of AMEISE developments.

The standard setup of these test runs is that after a brief introduction (90 min. lecture plus reading material), students get their first project assigned. Then, the results of the first assignment are discussed. Each group gets full written feedback and interesting aspects, notably second order effects, are discussed in front of the whole class. This gives students also a chance to ask questions about their own project when the handout is not sufficiently telling. After this feedback, students have a second chance to manage the project. This is important, since most of the students or groups fail to achieve all objectives, the simulated project should achieve. Severely exceeding development time or development budget are most prominent. Missing functionality (AFPs) or a fault rate beyond the level tolerated, are usual. The second run should give them a chance to perform better and hence help to engrain the experience gained.

The first of these "test runs" took place in the fall semester 2001/02. In this (elective) course, specifically dedicated to software project management, plain SESAM has been used. It helped us to review the aims of the AMEISE project. Further, we have seen that the pseudo natural language interaction provided by SESAM is not sufficiently user friendly. Students notably complained about case sensitivity and had problems with some rather clumsy names of tasks or documents. As result of this experience, the priority of redesigning the user interface has been raised.

Two further in-class tests of the system have been made in the spring semester 2002 and fall semester 2002/03. In both cases the system was used in a compulsory course on "System Development Process". For scheduling reasons, AMEISE was made available in the spring semester actually only as an addendum to the course early in the semester break. Nevertheless, all students (seventeen students in groups of two) took part in this offering and about half the class came back later to redo the assignment after a round of feedback by the instructor. The course of the fall semester had ten groups of pairs. Both simulation runs were made during the semester.

Both courses used already the distributed AMEISE architecture, but still the traditional SESAM user interface. The significance of these courses for the project was twofold. On one hand, it convinced us that the architecture achieving multi-user functionality by wrapping SESAM and interfacing this wrapper with a database keeping the state of simulation instances of individual students was feasible from a performance perspective. Had this test failed under real load conditions, different architectural solutions would have been needed. On the other hand, the behaviour of students, notably their mistakes in managing the assigned project were analyzed in depth. This provided important input to the design of the explanation component and of the support components consultant and friendly peer.

The experiences gained with these courses emphasized the requirement to develop a course- and model management component. This gave rise to a major architectural review of these components and resulted in the database design discussed in section 3.3 and the design of the specific aids sketched in section 4.3.

Another use of AMEISE was made in a course by the technical college (Fachhochschule Technikum Kärnten). Its major aim was to investigate the suitability of the concepts of this kind of project simulation by students that had followed a different educational path. The instructor of this course invested more time in the preparation than mentioned above. But overall, the educational results were similar to those of university students.

The most recent AMEISE course took place during the spring semester 2003, again as part of the class “System Development Process”. In this course, the new user interface (Fig. 4), the AMEISE evaluation component and initial versions of the consultant have been used for the first time. Students appreciated these features. Notably for the consultant (and by implication for the friendly peer) they made several suggestions for future extensions.

From the instructor’s point of view, one can say that the time it takes the students to simulate a project of 200 AFPs extending over a nine month development time was reduced from roughly four hours to about three hours. Thus, interaction with the system became markedly easier and students had to worry less about intricacies of the system but could better concentrate on the complexity of the actual project management task. It is up to the instructors though to make sure that simulations do not degenerate to a click-and-try venture. An important educational aim is to make students aware that managing a project of this size requires active planning and constant monitoring. Acting too short-sighted just on the feedback obtained from the system and its virtual developers will undoubtedly lead to missing the objective function to be reached by this simulated development.

On a meta-level, we may report that although students were aware that they are also partly acting as experimental subjects in these courses, they were quite excited about the experience gained with SESAM/AMEISE. Asked about the difference between having been told most of the things that were discussed in the intermediate feedback session in earlier software engineering classes and the discussions on the basis of the simulation runs, the standard reply was: “Now, these teachings are founded on (my) experience!” or: “But now, I not only heard you telling this, now I felt it!”

6. Future Developments

AMEISE is a still ongoing project. The features described here have been implemented and were explored in classroom situations as described above.

Our next steps with respect to implementation concern mainly the competitive features of group comparisons. This requires not only very careful consideration of the user interface but also explorations concerning those points in the individual projects, where comparisons are meaningful. Basically, points when milestones have been passed will be such “synchronization points”. However, this evident technical answer has to be weighted against the didactical question to which extent students are prepared to interpret intermediate results they obtain at these points.

This leads also to another important aspect of further work. As the features of friendly peer and consultant can be customized by the instructor, empirical work is still needed to determine in which situation what degree of support is most appropriate to obtain the didactical goals of the project.

We depart from the assumption that the various features implemented in AMEISE are not suitable for being exposed to students which are initiated to project management in a particular course. Doing so might degenerate AMEISE from an educational tool to a shiny simulation toy. Instead, instructors have to plan carefully what particular aims they want to achieve in their course and customize the tool’s features appropriately. Part of this customization concerns also the simulation model itself. A quick solution in this respect was that we implemented the possibility to outsource certain tasks to a software house. This reduces the overall complexity of the task to be managed by students. To explain this in detail would, however, require an in-depth discussion of SESAM models. This is beyond the scope of this paper.

7. Conclusion

AMEISE provides non-standard educational software in so far, as students should get an overall impression of the complexity of project management by managing a set of virtual individuals in a rather tightly budgeted project. In contrast to majority of e-learning materials, AMEISE does not beef up textbook like information by animation and interaction. It gives students the chance to reflect on their own performance by giving them specific feedback concerning their behavior in solving a multi-criteria optimization problem with partly conflicting sub-objectives.

The project builds on software and experience developed at the University of Stuttgart. This allowed us to experiment while developing the various additions to the original SESAM software. Our initial experience shows that students do appreciate the experience they gain with this simulation tool. Further, the enhancements provided by AMEISE help them to focus more on the real task than on intricacies of the system and to obtain fast feedback. They also allow instructors to customize software project simulation according to their specific educational aims.

References:

[1] Drappa, A.; Ludewig, J.: Quantitative Modeling for the Interactive Simulation of Software Projects. The Journal of Systems and Software 46, April 1999, P.113-122
[2] John Barnes: Programming in ADA 95. Addison-Wesley Publishing Company. 1998.
[3] John K. Ousterhout: Tcl and the Tk Toolkit. Addison-Wesley Professional Computing Series. 1994.
[4] Drappa, A.: Quantitative Modellierung von Softwareprojekten. Doctoral Dissertation. Shaker-Verlag, Aachen, 2000.
[5] Nusser M.: Projektunterstützung durch Verlaufsanalysen und Agenten. Universität Klagenfurt, Institut für Informatik-Systeme, 2003.
[6] Jäger S.: Entwicklung eines elektronischen Tutors zur Analyse von Projektverläufen. Universität Klagenfurt, Institut für Informatik-Systeme, 2003.

Authors:

Roland Mittermeir, O. Univ. Prof. Dr.
Institut für Informatik-Systeme, Universität Klagenfurt
Universitätsstraße 65-67, A-9020 Klagenfurt
roland@isys.uni-klu.ac.at

Elke Hochmüller, FH-Prof.. Dr.
Telematik/ Netzwerktechnik, Fachhochschule Technikum Kärnten
Primoschgasse 8, A-9020 Klagenfurt
e.hochmueller@fh-kaernten.at

Andreas Bollin, DI
Institut für Informatik-Systeme, Universität Klagenfurt
Universitätsstraße 65-67, A-9020 Klagenfurt
Andreas.Bollin@uni-klu.ac.at

Susanne Jäger, DI
Institut für Informatik-Systeme, Universität Klagenfurt
Universitätsstraße 65-67, A-9020 Klagenfurt
Susi@isys.uni-klu.ac.at

Markus Nusser, DI
Telematik/ Netzwerktechnik, Fachhochschule Technikum Kärnten
Primoschgasse 8, A-9020 Klagenfurt
m.nusser@fh-kaernten.at