

A Framework System for Intelligent Support in Open Distributed Learning Environments

M. Mühlenbrock, F. Tewissen, H.U. Hoppe, *University of Duisburg, COLLIDE Research Group, Dept. of Math./Computer Science, 47048 Duisburg, Germany*

Abstract. Recent trends in the design of learning support systems are characterized by considering group interaction, by combining intelligent support with interactive learning environments, by providing reusable domain-independent components, and by using agent-structured architectures. Taking these trends into account, an open framework system has been developed for integrating distributed intelligent support components with an interactive and collaborative learning environment. Work in shared activity spaces with synchronized objects is facilitated by a specific user interface library and communication server. A modular architecture allows for flexibly plugging in intelligent components, e.g. for knowledge assessment and diagnosis, individual feedback, multiple student modeling, or simulated students as a partner. The intelligent subsystem is organized as a cluster of student modeling agents and is inspectable through a WWW interface. Our approach is exemplified by several applications, one providing individual feedback for physics exercises, a second one supporting the teacher in supervising a group of students in a math class, and, thirdly, a generic tool based on card networks for diverse domains.

INTRODUCTION

Over the last couple of years we have experienced a paradigm shift in innovative learning support systems. AI in education is particularly confronted with (and must react to) the following new trends:

- consideration of group interaction, i.e. design for multi-user learning environments,
- combination of intelligent learning support with interactive learning environments,
- provision of intelligent components or plug-in modules rather than all-embracing systems based on specific AI frameworks
- use of the multi-agent paradigm for the system architecture.

We feel that the first three trends designate a sort of realistic turn that reflects a more general tendency towards combining AI-based systems with standard software tools and architectures. The multi-agent paradigm is of more conceptual nature. It has certainly introduced new ways of thinking and speaking about systems and particularly about architectures. However, it needs to be further substantiated, at least in our field, in terms of bringing forth new types of systems. In this paper, we describe the concept and the current version of a general framework system that fits in with these trends. Our approach is consistent with current efforts to develop “architectures and methods for designing cost-effective and reusable ITS” (Suthers 1996).

In our understanding, the term *open distributed learning environment* (ODLE) is a suitable common denominator for the new orientation of educational systems. It integrates the ideas of interactive learning environments and of collaborative learning, cf. Dillenbourg, Baker, Blaye & O'Malley (1995), supported by distributed computing and communication facilities, as well as the idea of openness in terms of using standards and modular plug-in architectures. The specific theme of this paper is the development of a system platform that is suited for integrating intelligent support modules as agents within a general ODLE framework.

Concerning the role of AI-based components, we start from the premise that there is rich body of existing AI technology for educational use that, rather than becoming obsolete through the adoption of new paradigms, may be used much more practically as modules inside new ODLEs. This is particularly true for diagnostic algorithms and student modeling techniques that may serve as analytic background mechanisms to assist teachers, human tutors, and students.

Our practical experience with ODLEs is centered around two different scenarios: (1) the computer-integrated classroom (CiC) with a large interactive display and with student computers integrated in a primarily local network (see figure 1), and (2) the facilitation of peer help and tutoring on demand in a virtual group of networked learners. *Multiple student modeling* (Hoppe 1995), defined as an extension of standard methods for individual modeling, can be used as a support module in scenario (2). In the first scenario, support can be given as individual feedback to students or groups working on exercises, but also to the teacher in the form of an intelligent aid for supervising and organizing the learning group. Our framework system provides a uniform platform for both kinds of applications.

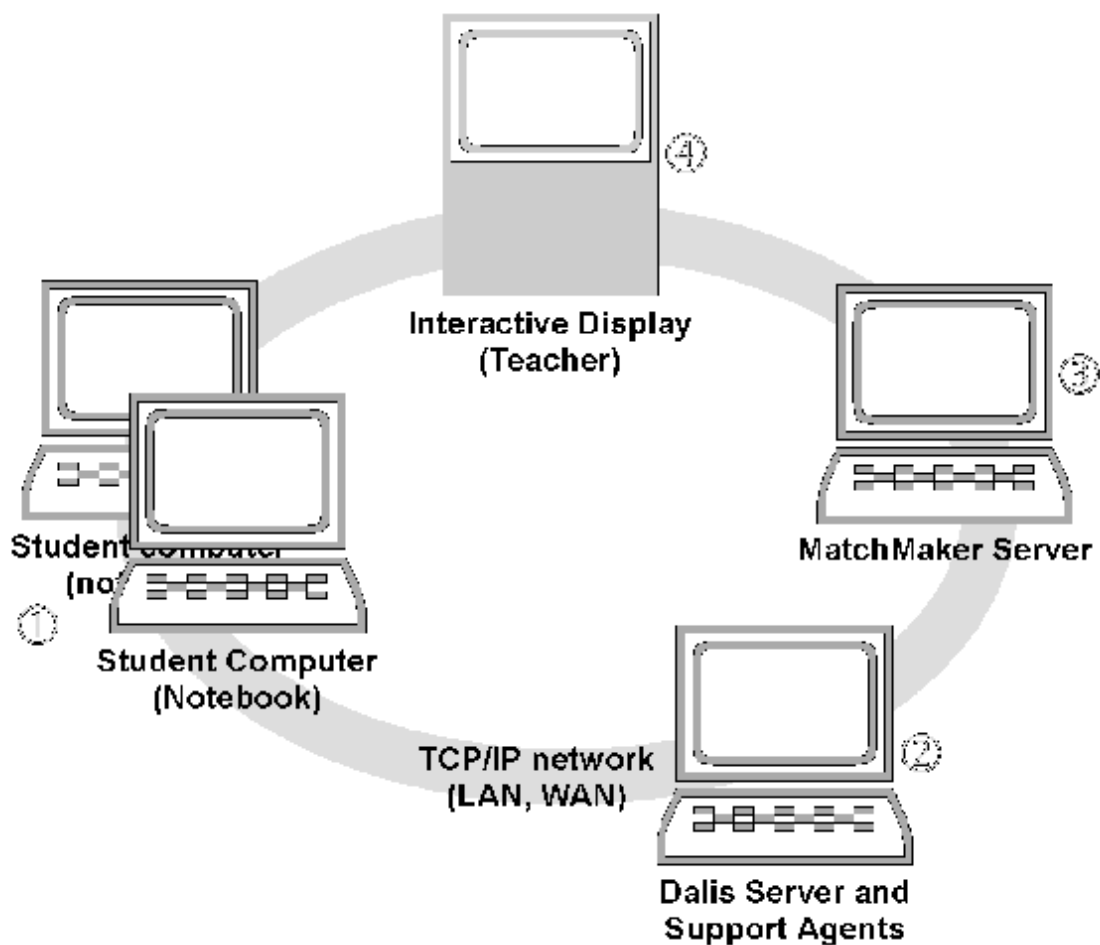


Figure 1. CiC environment

COMMUNICATION AND COOPERATION MECHANISMS

In both of the mentioned ODLE scenarios, there is communication between humans (either natural or technically mediated), between humans and the system, and internal system communication. In the CiC, natural communication is both possible and useful and should only be supported and enriched by technical means. In the virtual scenario (2), any human-human communication is necessarily technically mediated. Technically mediated human-human

communication has two aspects: telepresence and joint actions on shared objects or shared workspaces. In our current system environment, telepresence is provided by separate tools (ISDN or LAN video-conferencing systems), but shared workspace operations as well as internal system communication are facilitated by the architecture in a uniform way.

We believe that *shared activity spaces* are an important element of collaborative learning environments (Plötzner, Hoppe, Fehse, Nolte & Tewissen 1996). By co-operating simultaneously or by following adequate turn-taking rules, individual contributions are objectivated and externalized in persistent form as compared to the volatile nature of spoken language communications. The material used by students can also be of abstract, symbolic nature (e.g. formulae or abstract diagrams). The epistemological foundations and implications of communication and cooperation through shared actions are of general interest for theories of collaborative learning. In this paper, we will only discuss technical prerequisites for further work in this direction.

All our ODLE applications (cf. section 5 and 6 for examples) comprise an environment or microworld with more or less rich interaction that can be used in both shared and single-user mode. Sharing is technically facilitated by a general mechanism of *coupling* or synchronizing user interface (UI) objects. This UI coupling service is supported by one of two central communication servers, the MATCHMAKER server (see item (3) in figure 1). It allows user interfaces to share information and synchronize actions on UI objects, and it also supports remote procedure calls for internal communication. The object-wise coupling method is very flexible in that it allows for dynamic coupling between objects with potentially different presentation types. The application programmer does not deal with the lower levels of the communication protocols, and the communication through the provided high level message primitives only needs a small network bandwidth. This UI part of an application, including the interactive environment and the coupling mechanism, is self-contained and functional without intelligent support. Both this communication mechanism as well as the multi-agent architecture for intelligent support (to be described in section 4) make use of the standard TCP/IP protocol and provide abstracted programming interfaces.

The communication model and the principle of communication with and through coupled UI objects are originally defined in Zhao & Hoppe(1994). The communication facilities are built into a UI library of object types and functions called MATCHMAKER, which is currently implemented as a C++ library. Additional programming for enabling the coupling of UI objects is not necessary. Every MATCHMAKER-based user interface exists as an autonomous application in a replicated architecture. This communication design gives the application programmer a clear conceptual model of the mechanism of object coupling; if two or more UI objects are coupled, all events reaching one of these objects are broadcasted to all the other objects. For instance, pressing a coupled button in one user interface produces the same effect as pressing decoupled buttons in every user interface.

Table 1 shows a list of all communication-related MATCHMAKER methods. The methods are separated into three groups. The first group covers the methods related to coupling. These methods initialize the coupling of UI objects or of children objects, dispose the UI object coupling and distribute messages via the MATCHMAKER server that keeps track of every change of the coupling-database. The second group comprises three ways of executing remote procedure calls: Calling the callback-methods of a named object, calling a named method directly and providing an interface for communicating with external (non-MATCHMAKER) processes (e.g. to provide access to the DALIS server and support agents, see item (2) in figure 1). Group 3 deals with services provided by the server: access to the property-databases of the applications and a few general services (e.g. generating unique identifiers). The most interesting method might be the method MmCouple. Once the objects have a symbolic name they can simply be coupled with this method. The initialization, synchronization and the life-cycle handling of coupled objects are facilitated by the central server and the underlying local MATCHMAKER functionality.

Table 1: MATCHMAKER Communication Primitives¹

Method	Arguments	Description
MmCouple	host1, app1, obj1, host2, app2, obj2	Couple object //host1/app1/obj1 with object //host2/app2/obj2, obj1 initially overwrites state of obj2. Both objects may be local or remote.
MmCoupleChildren	obj1, host, app, obj2	Couple all child objects of the local parent obj1 with children of parent //host/app/obj2.
MmDecouple	host, app, obj	Decouples object //host/app/obj.
MmCreateCallback	obj, msg, fmt, data	Creates a callback message with the arguments msg, fmt and data on object obj and distributes it to all objects coupled with obj.
MmSendFunctionMessage	host, app, fnc, msg, fmt, data	Executes a remote procedure call on the function //host/app/fnc with the arguments msg, fmt and data.
MmSendObjectMessage	host, app, obj, msg, fmt, data	Executes a remote procedure call on all callback functions of object //host/app/obj with the arguments msg, fmt and data.
MmSendSocketMessage	host, port, fnc, msg, data	Sends the byte stream message data to //host:port. Results are send back to function fnc labeled as msg.
MmServerService	srv, fnc, fmt, data	Requests a common server service srv with the arguments fmt and data. Results are send back to function fnc.
MmSetAppProperty	prop, val	Sets an application property prop with the value val for the local application in the server's database.
MmGetAppProperty	host, app, prop	Gets the value of the application property //host/app/prop from the server's database.

STUDENT MODELING

The online analysis of group interactions aiming at assessing and updating individual learner models of the group members appears to be currently intractable for unrestricted interaction, including (spoken) natural language or even non-verbal communication. Even if the group interaction leads to concrete actions and state changes in shared workspace environments, it is often not evident how to assign individual credits for concrete achievements or failures. Following a recent suggestion by Paiva (1997), this difficulty can be avoided by treating the entire group as a subject of the modelling process according to the principle that *the group is more than the sum of its parts*. In the approach of Paiva (1997), the group model should be built upon actions and beliefs on which the group members have agreed. However, with unrestricted interaction, the tractability problem is now shifted towards defining operational criteria for agreement. Also, it is not clear how to initialize the group model when the collaborative learning session starts after a phase of individual work.

¹ The URL syntax //a/b/c[.d]* is used to mark an entity that is registered with the symbolic name c[.d]* in application b that is running on host a.

We believe that the general problem of analyzing rich group interactions will be an issue of research for the next years to come, and it may even turn out that certain aspects of the problem are indeed intractable. To avoid the complexity of analyzing realistic group interactions, we suggest to start with using individually assessed student models to *parameterize* group learning situations (Hoppe 1995). To make this more modest approach widely applicable, it is desirable that the extrapolation of individual student models not be dependent on a special modelling technique. So, we depart from general information that is essentially provided by any type of student model. Accordingly, we do not make specific assumptions about the actual assessment of the model, i.e., about the diagnostic procedure.

Though learner modeling for groups of learners introduces a new type of complexity, the availability of human support, such as peer-to-peer cooperation, can also help to avoid existing problems in generating adequate, personally meaningful feedback to learners. Generally speaking, intelligent subsystems may support the tasks of knowledge assessment and error diagnosis, whereas the actual tutoring may be left to human-human interaction, be it direct or technically mediated. While in the presence of a teacher or human tutor, individual problem solving phases (exercises) are typically only monitored and analyzed locally and feedback is given directly to the teacher, unmonitored situations require an integration of the multiple student models to infer the adequateness of cooperation between certain individuals.

A general conceptual and formal framework for student model integration is introduced in Hoppe (1995) under the notion of *multiple student modeling*. The general premise is that individually assessed learner models can be used to support the configuration or parameterization of collaborative learning settings. These are prototypical cases:

- Given a number of students working on comparable problems in an open learning network, find pairs of students that could potentially benefit from cooperation in a joint session. The selection can be based on such criteria as complementarity or competitiveness.
- Given a group of students, select or generate a problem that forms an adequate challenge for the group as a whole. The problem should not be solvable by one student's knowledge alone, but rather through the union of all the students' individual knowledge bases. In this case, the challenge for the group consists in knowledge exchange and integration.

Selection criteria for these prototypical cases can be formulated on the basis of general modeling primitives such as `knows(Student,Topic)` or `has_difficulty(Student,Topic)`, which can be inferred from different standard types of student models. A simple case of knowledge integration is exemplified by the rule

```
can_help(Student1,Student2,Topic) <--
    knows(Student1,Topic) &
    has_difficulty(Student2,Topic).
```

This rule is actually used in a peer helper scenario. Accordingly, the architecture of the intelligent subsystem must allow for combining elements from different individual student models. In the original example, individual diagnosis did not require backtracking and modeling was cumulative for all learners at a time. However, diagnosis with backtracking and user interaction needs a more flexible, parallel or multi-threaded architecture. Such an architecture will be introduced in the next section.

Individual tutoring ideally takes place in a *closed loop* in which every utterance or action of the tutee is interpreted and can lead, if considered useful by the system, to specific feedback. This is practically not feasible with rich group interactions. Thus, in intelligently supported group learning we do not expect to have a closed feedback loop but an open system that evaluates and intervenes only under certain specific conditions. This general framework is visualized in Figure 2.

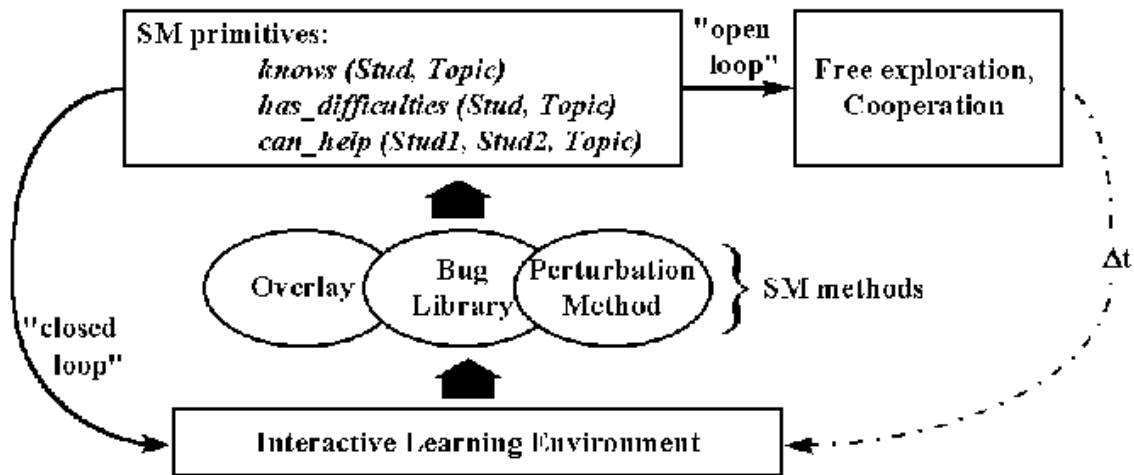


Figure 2. Open framework for student modeling, from Hoppe (1995)

Open-loop learning with a learning support system is indeed not only specific to group situations. It is also typical for free individual exploration in interactive learning environments (ILE) or microworlds. Here, the role of the environments consists in providing a rich *activity space* that rather implicitly than explicitly creates opportunities for learning. However, if given a limited, partial function in evaluating and supporting the learning process, student modeling can also play a role in ILE (Self 1994). Phases of more or less free or unguided exploration may e.g. be followed by exercises with closed-loop feedback.

There is a wide range of different support functions that can be implemented in such a modeling framework:

- *intelligently mediated peer help*
Here, the individually assessed learner models are used to match pairs of learners that should maximally benefit from each other when working together. The prediction can be based on different criteria such as complementarity of skills/knowledge or competition. A first implementation of this principle has been reported in Hoppe, Baloiian, Schupp & Zhao (1995). Massive practical applications of a similar type have recently been described by McCalla, Greer, Kumar, Meagher, Collins, Tkatch & Parkinson (1997).
- *intelligently mediated expert tutoring*
Formally, this case can be considered as a specialization and simplification of matching peer learners, since only one of the models (the learner's) has to be dynamically assessed, whereas the tutors' profiles may be predefined.
- *teacher/tutor support for supervising individual exercises*
Essentially a decision support function for the teacher. To achieve this it is sufficient to aggregate the individual learner models in a form that allows for filtering out specific features, e.g. frequent problems. The support mechanism can also actively inform the teacher if adequate.
- *group formation around given problems*
This is a generalization of mediating peer help in that the number of group members is not restricted to two. Also the problem requirements must be analytically specified.
- *selection of adequate problems for a given group*
A problem is e.g. selected or generated in such a way that it could serve as a challenge to the group as a whole but should still be feasible if the group were able to combine individual strengths.

INTELLIGENT SUPPORT

User and group assistance in ODLEs call for a simple yet powerful mechanism to extend the user interfaces with intelligent components, thereby reacting to the needs of the various learning and collaboration modes in a flexible and scalable manner. The development of the intelligent support system is based on the following goals:

- a simple *plug-in* connection to any ODLE as described in section 2,
- compatibility with a wide range of standard user modeling techniques with no assumptions about the specific nature of the model, and
- design for collaborative, multi-user environments.

Above all, these underlying assumptions speak for a flexible, parallel or multi-threaded support component, since multiple user requests for advice and collaboration have to be processed concurrently.

It has been argued that for ITS to achieve more general and reusable learner models, learner modeling systems have to be decoupled from the other software parts, and considered as a kind of software agent (Paiva 1996). In our approach, we have adopted the notion of a software agent in a broad sense; i.e. that some entity is an agent if it communicates correctly in a specified communication language and if it is constrained by behavioral principles such as autonomy and independence (Genesereth, Singh & Syed 1994). This is comparable to the KQML approach (Finin, Fritzon, McKay & McEntire 1994; Finin, Labrou, Mayfield 1997) and its incorporation in learner modeling systems such as e.g. the system GIA by Cheikes (1994). However, the goal is not to build another general multi agent system, but to provide a framework system that is particularly suited for flexibly plugging support agents into existing ODLEs. This approach is eased by the open structure of ODLEs which helps to avoid many of the difficulties that Ritter & Koedinger (1996) had to tackle in their approach of plugging-in tutoring agents into rather closed pre-existing programs.

The intelligent support system DALIS (Distributed Architecture for Learning with Intelligent Support) is centered around a message server (see item (2) in figure 1) that provides anonymous interaction between agents on the basis of a specific communication protocol. The agents relinquish part of their autonomy to the DALIS server, which therefore allows for a uniform user interface and agent administration. This corresponds to a federated system headed by a facilitator (Genesereth 1997). A difference is that the DALIS server also facilitates entities outside the federation architecture, i.e. the user interfaces, in order to comply with the first design goal (see above).

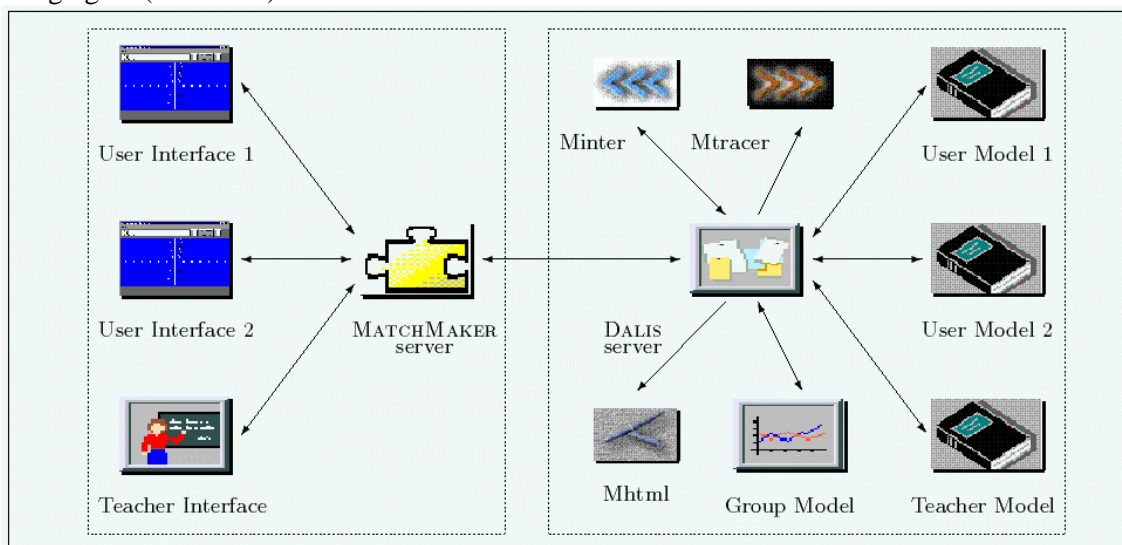


Figure 3: Sample configuration of the framework system based on the MATCHMAKER and DALIS servers

Figure 3 shows on its left a sample configuration of a MATCHMAKER centered ODLE with some user interfaces and a teacher interface as described in section 2. In the figure, the ODLE has been extended by the DALIS agent environment for individual and group support (see figure 3). Its architecture is based on the design principle of *mirroring* the topology and interactivity of the real learning group by a Prolog-based monitoring system composed of individual agents. The agents have in common a standardized communication interface to the DALIS server. The entire Prolog functionality including backtracking can be accessed through this interface. By using this general mechanism, it is quite simple to add arbitrary Prolog programs as new agents to the environment.

In order to extend an ODLE by components for individual and group support, the user interfaces first have to register with the DALIS server (facilitator) by way of the MATCHMAKER server. The user interfaces send information about their specific types and their languages to the DALIS server. For each newly registered application, the DALIS server invokes a pre-specified set of support agents, which again register with the server in the same way as the user interfaces. Some support agents are only invoked once and their services are shared among other agents and user applications. A typical example for a shared agent is the group monitoring agent in a learning environment or an artificial player for a game-like application. There are also some additional shared monitoring agents for system maintenance and WWW interfacing, which will be described below.

Table 2: DALIS communication and specification primitives

Functor	Arguments	Description
init	<i>AgentType</i> <i>AgentLanguage</i>	Initialization message used by a user interface or support agent to register with the DALIS server; the server builds communication clusters based on the <i>invoke</i> and <i>share</i> specifications (see below)
inform	<i>AgentMessage</i>	message send to user interfaces or support agents
do	<i>AgentMessage</i>	query message send to agents; allows for backtracking on query (see below)
redo	<i>AgentMessage</i>	triggers backtracking on message
fail		failure an evaluating previous message; no results given or no more backtracking possible
invoke	<i>AgentType1</i> <i>AgentType2</i>	Registration of a user interface or support agent with type <i>AgentType1</i> triggers the invocation of a support agent with type <i>AgentType2</i> by the DALIS server; the agent with type <i>AgentType2</i> is grouped in the cluster of the agent with type <i>AgentType1</i>
share	<i>AgentType1</i> <i>AgentType2</i> <i>AgentTypeShared</i>	user interfaces or support agents with types <i>AgentType1</i> and <i>AgentType2</i> share agents with type <i>AgentTypeShared</i> ; the latter are only invoked once for newly registered agents with types <i>AgentType1</i> and <i>AgentType2</i>

The communication between agents is based on a small set of message primitives. The user interfaces and the agents register with the DALIS server by the primitive *init* to inform the server about their types and languages (see table 2). The agents' types determine the invocation of other agents and sharing of agent services. The automatic invocation and sharing of agents can be pre-specified with the DALIS server by the primitives *invoke* and *share*, respectively. Based on this specification, the user interfaces and their corresponding agents are grouped in communication clusters, in order to prevent interference of messages from agents or interfaces of the same types. However, multi cluster membership is possible for shared agents. The

language information that an agent sends with its registration provides an implicit addressing scheme by determining the dispatching of messages by the server to the connected agents within a cluster.

Queries are initiated by the primitive `do`, and `redo` resumes backtracking on the previous query. Messages labeled with the primitive `inform` are similar to `do`-messages, but do not give the possibility of backtracking. If a query fails or no more backtracking is possible, the agent will send back the fail primitive. These communication primitives are comparable to a subset of the KQML agent communication language. They correspond to the performatives `register`, `generate` (short for `standby with content stream-all`), `inform`, `next`, and `sorry` in KQML. Figure 4 shows a sample message flow between user interfaces, servers, and agents during a process of agent invocation and interaction. In this example, the registration of `UserInterface1` leads to the invocation of `UserModel1` and `GroupModel`, and the latter two placed in the communication cluster of `UserInterface1`. The agent `GroupModel` would be shared with further user interfaces and user models.

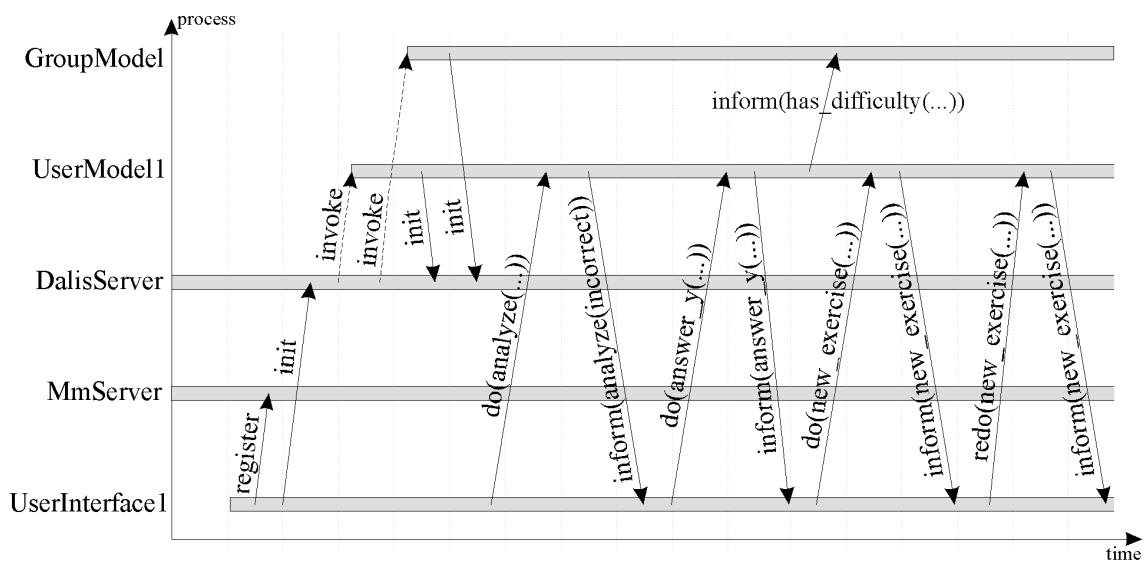


Figure 4: Message flow between user interfaces, servers, and agents

The modeling system is also combined with some standard tools: As the most prominent feature, dynamically updated HTML files are used as a means for inspecting the flow of information between the user interfaces and support agents that have registered with the DALIS server. This is particularly useful on a heterogeneous system platform with no common file system shared by the two components but only a TCP/IP-based Internet connection.² In combination with a standard WWW sever, the information generated can be easily accessed from each system platform. The HTML component generates an overview of the currently registered agents and user interfaces (see figure 5), and for each of these components there is an additional linked page that shows received and sent messages (see figure 6, which includes part of the sample communication outlined in figure 4 from the point of view of a user model, with the latest messages on top of the list). Though it is currently used for system testing and development, it is a potential platform also for educationally oriented supervision tasks and for the transcription and monitoring of experiments.

² In our environment, the user interfaces (for teacher and students) run on Windows PCs, whereas the modeling subsystem runs on SUN workstations or PCs.

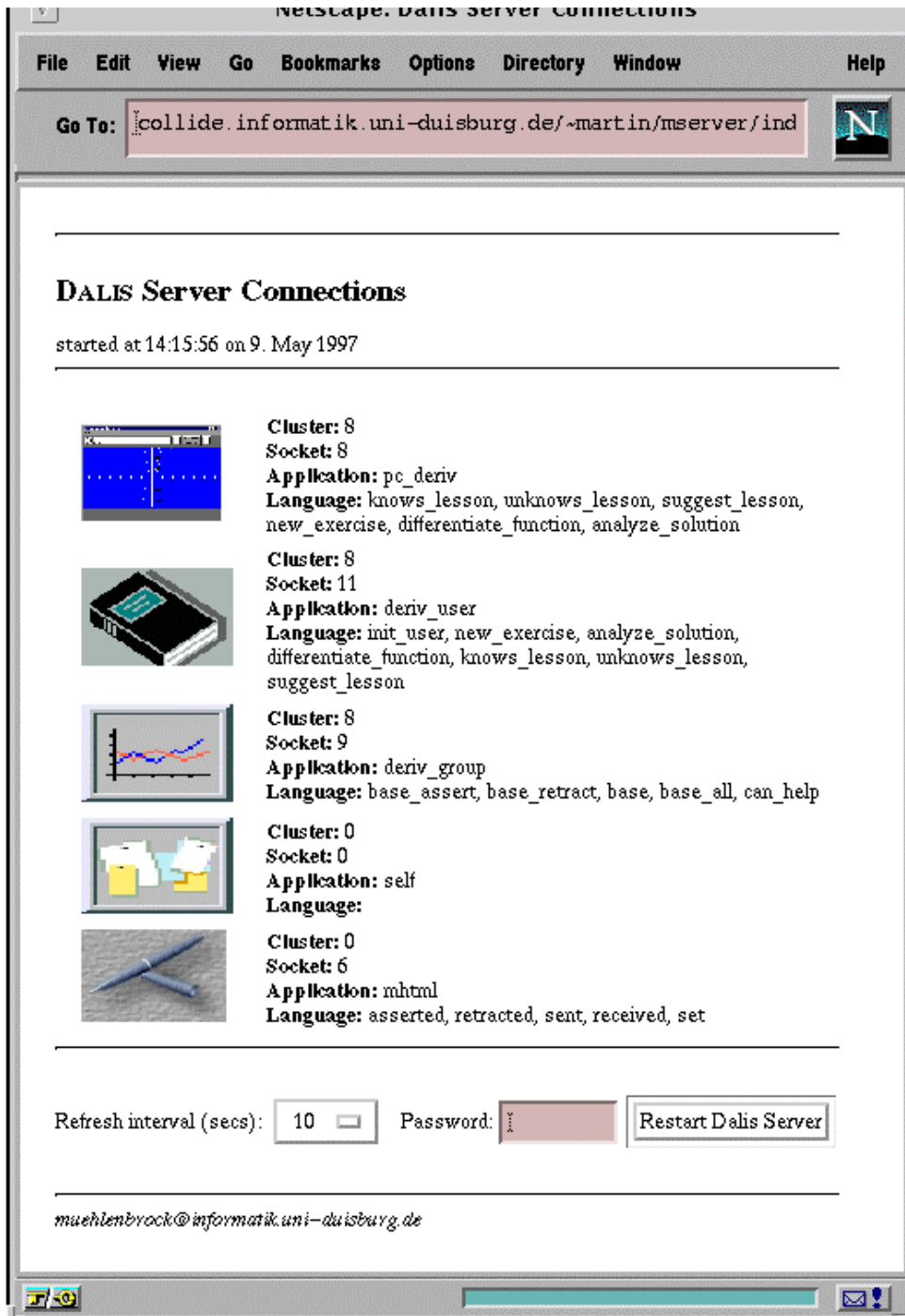


Figure 5: Server connections

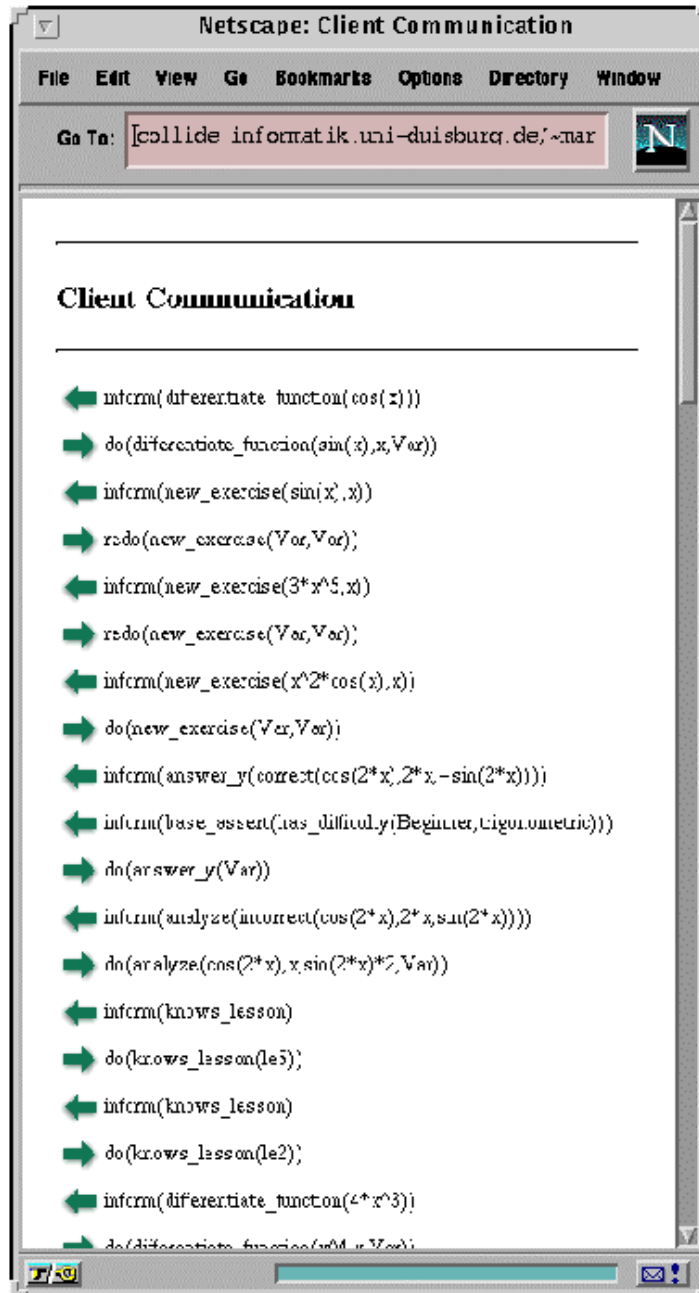


Figure 6: Client communication

DOMAIN-SPECIFIC APPLICATIONS

On the basis of the MATCHMAKER and DALIS framework system, several prototypical applications have been developed, e.g. in the domains of high school mechanics (*CardMan*) and symbolic differentiation (*SuperDeriv*). The applications differ along the following lines: On the one hand, whether they provide domain-dependent user interfaces or employ a more generic interface that can be instantiated for a range of domains. On the other hand, whether the intelligent support component is realized as a coaching system or as a computer partner such as

e.g. an artificial game player. The early systems *CardMan* and *SuperDeriv* come with domain-dependent user interfaces and coach the student in solving exercises.

In the shared workspace environment of *CardMan* (Tewissen 1996) for the domain of high school mechanics, all terms and operators, but also diagrams and other graphical material, are uniformly represented as “cards”. The notion of a card has been derived from a psychological study with real paper material (Plötzner et al. 1996). With these cards, the students can arrange card nets of their solutions within their private workspaces. They can also cooperate with each other in shared workspaces. Once a problem or a question arises, the students can help each other or they are supported by the system through locating likely mistakes or referencing relevant course material. This application for high school mechanics served as a basis for the development of a more domain-independent application framework that will be described in the following two sections.

Whereas *CardMan* provides individual feedback and context-sensitive suggestions on demand, the application *SuperDeriv* (Supervisor for Derivations) additionally uses individual error analyses to support the teacher in supervising a group of students working on exercises (Mühlenbrock, Tewissen & Hoppe 1997). In this application, the communication architecture has been employed for providing intelligent support not only for individuals but also on the group level. The diagnostic procedure follows a reconstructive approach, namely *deductive diagnosis* (Hoppe 1994). Given a correct and complete domain theory, in this case symbolic differentiation, an incorrect student solution is indicated by an unprovable goal in the process of reconstructing a correct solution using a fail-safe meta interpreter. In the course of backtracking on different sets of error conditions, the algorithm determines structural conditions of erroneous examples. In the context of group learning, the individual student models are accumulated and integrated to derive a model of group problem solving that initiates and supports remedial activities.

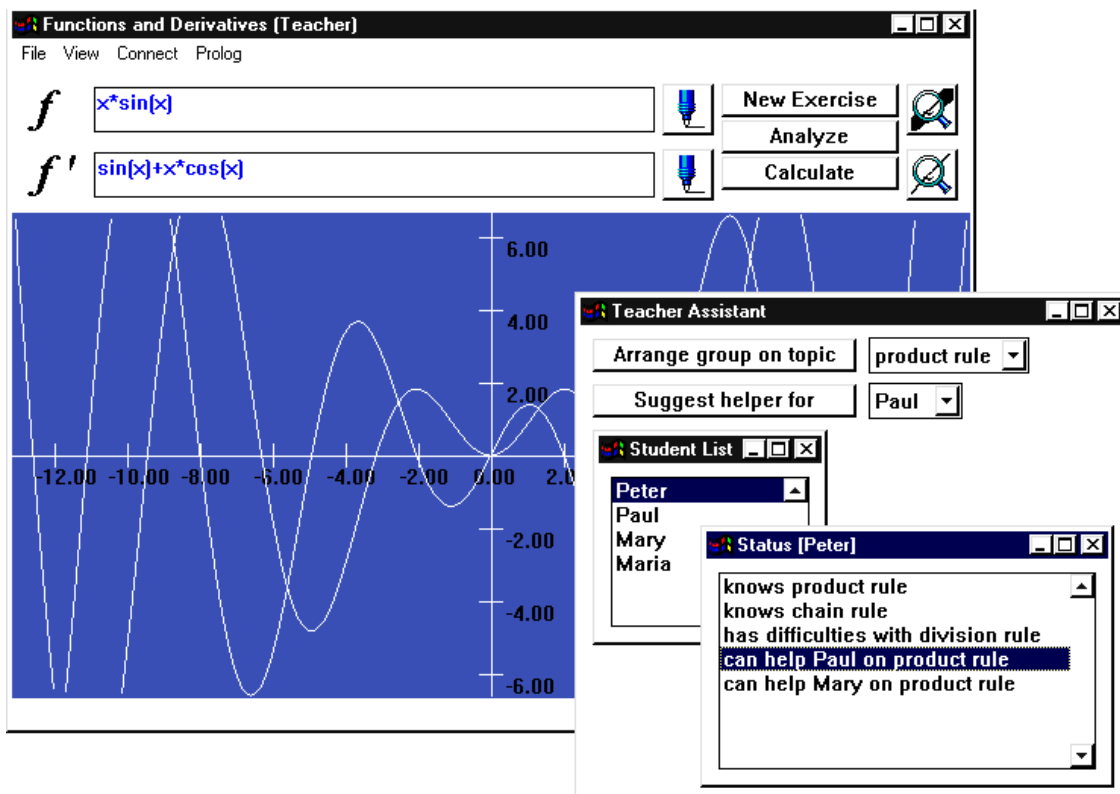


Figure 7: *SuperDeriv* teacher interface

Since the derivation tutor has been implemented in Prolog, it was easy to integrate the tutor into the DALIS architecture. Here, the backtracking facility of the architecture (cf. section 4) plays an important role for the generation and testing of error hypotheses. For each student, an

agent is automatically invoked which keeps track of the student model and is connected to a group monitor that integrates the individual student models. Figure 7 shows the components of the teacher interface: the exercise window prepared for coupling with an equally structured student interface, and a tentative version of a teacher assistant, which shows the output of the group monitoring agent. It allows for inspecting a student's current status concerning his knowledge and difficulties, for arranging groups on selected topics and for suggesting a appropriate peer helper for a student with difficulties.

GENERIC APPLICATION FRAMEWORK

The two applications mentioned in the previous section are domain-dependent to a considerable degree. Reusing these applications in domains other than high school mechanics and symbolic differentiation would require quite a bit of re-implementation, mostly on the part of the user interfaces. In order to overcome this limitation, the *CardMan* application has been generalized to a generic "card board" user interface (Gassner, Tewissen, Mühlenbrock, Loesch & Hoppe 1998).

The main features of the application *CardBoard* are cards, which generally speaking are containers for textual or graphical material, and workspaces with networks of cards. There are two kinds of cards, namely content and connector cards. Only connector cards can establish links to other cards. A link may be labeled and can be formed to any other card within the same workspace. Certain restrictions on cards concerning their default content and the number and labels of links of connector cards can be specified in separate configuration files. These are consulted at runtime. This specification is the only domain dependent part of a user interface and the separation of user interface and card specification allows for an easy adaptation to different domains.

A single *CardBoard* can contain several workspaces, and cards can be freely moved and copied between workspaces by drag and drop operations. Workspaces can be shared with other users by the synchronization mechanism of the MATCHMAKER server(cf. section 2). As graph-based representations have clear advantages in representing structural, logical or temporal dependencies in many formal and semi-formal domains, a special component *CardDalis* has been developed for the provision of operational semantics for card nets.

For the generic application framework *CardBoard* and *CardDalis*, we have developed a specific communication protocol to define messages on the creation, deletion, and modification of applications, workspaces, cards, and links. Every user action is reported in terms of this communication language to every coupled application and to the *CardDalis* agent. The *CardDalis* component keeps track of the changes in the workspaces and reconstructs a model of the workspaces and their contents. The agents can equally initiate modifications of workspaces by the same set of message primitives. In this way, the support component can communicate with users in the same way as users communicate with each other in shared workspaces, i.e., by creating or deleting cards or modifying card nets. The benefit from a systems engineering point of view is that additional functions to the support component can be integrated into the user interface not by re-designing the interface but by extending the corresponding card specification.

Operational semantics for card nets is provided by integrating appropriate interpreters in *CardDalis* and defining a translation between the descriptive representation of the card nets and the representational structures of the particular interpreter. A set of topological and graph-based abstractions for workspaces is already defined. In the sample application in figure 8, the cards are specified in such a way that they allow for simple deductive queries over a relational knowledge base. There are content cards for constants and variables and connector cards for relations and logical combination of relations (unit clauses). Here, the task is to create a relational model of some domain (e.g. family relationships) and to verify the model by posing queries against relations previously defined. In this case, the underlying semantics is simply the standard Prolog interpretation. However, any other interpreter that is implemented in Prolog can be incorporated in the framework system. For instance, the arithmetic interpretation within

SuperDeriv (cf. figure 7) can be integrated in the same way, as will be described in the next section.

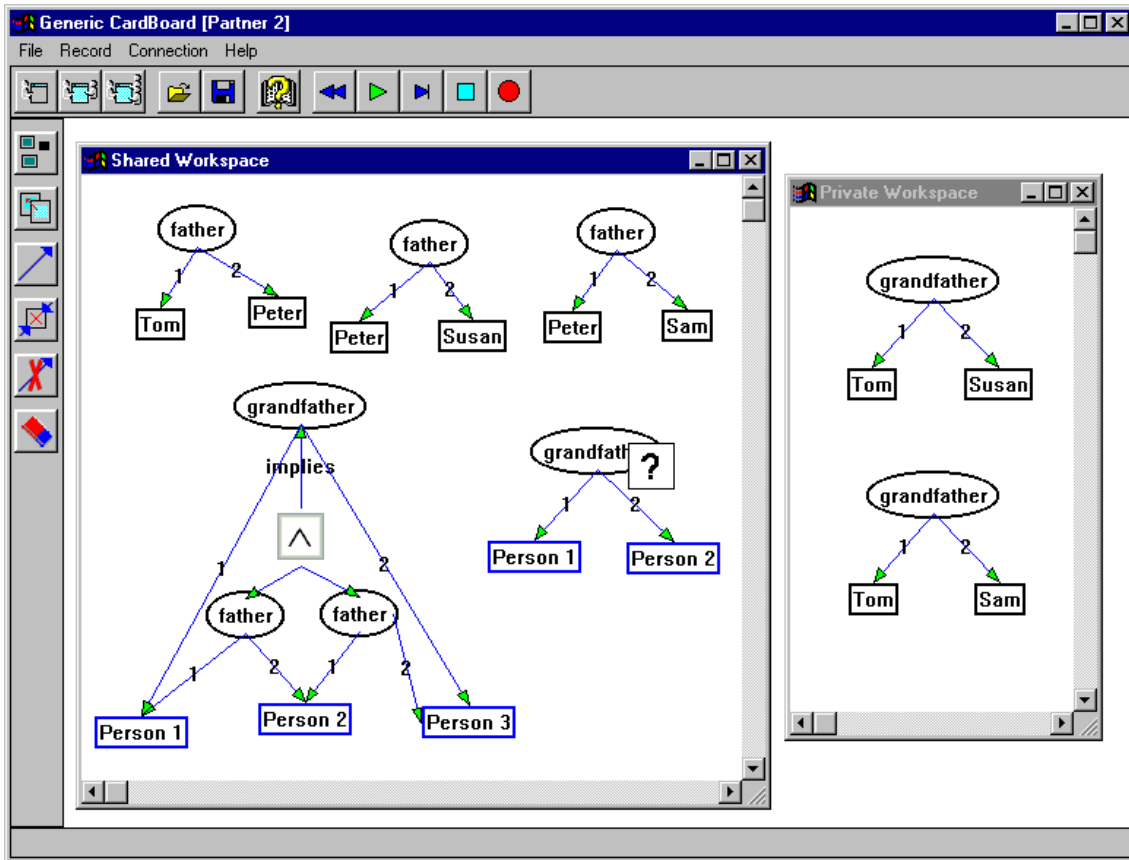


Figure 8: CardBoard interface for family relationships

To initiate and request context-dependent support functions, a small set of request cards such as a question and an exclamation card are given. Primarily, they are defined for human-computer communication, but they can also be used for human-human communication. They obtain their meaning through the context of the workspace and the cards they have been placed on. For instance in the example shown in figure 8, placing a question card on a connector card will be interpreted as a query to the current database. In the game-like application in figure 9, the question cards are used to ask the computer player to add one of its individual turtle cards to the puzzle in the central workspace. This application has been developed to introduce students to the principles of collaboration in shared workspaces. The set of cards with turtle symbols is split up among the participating players. In case there are not enough players, *CardDalis* takes over the role of an artificial player with the left over cards. The task is to collaboratively join the cards to form a coherent picture of turtles with matching colors and shapes. The request to the artificial player is based on an inferred adjacency relation between cards. After a request to the artificial player, the question card is deleted. Since there is only a limited number of question cards available for each player, the players have to carefully think about where and when they should use their question cards.

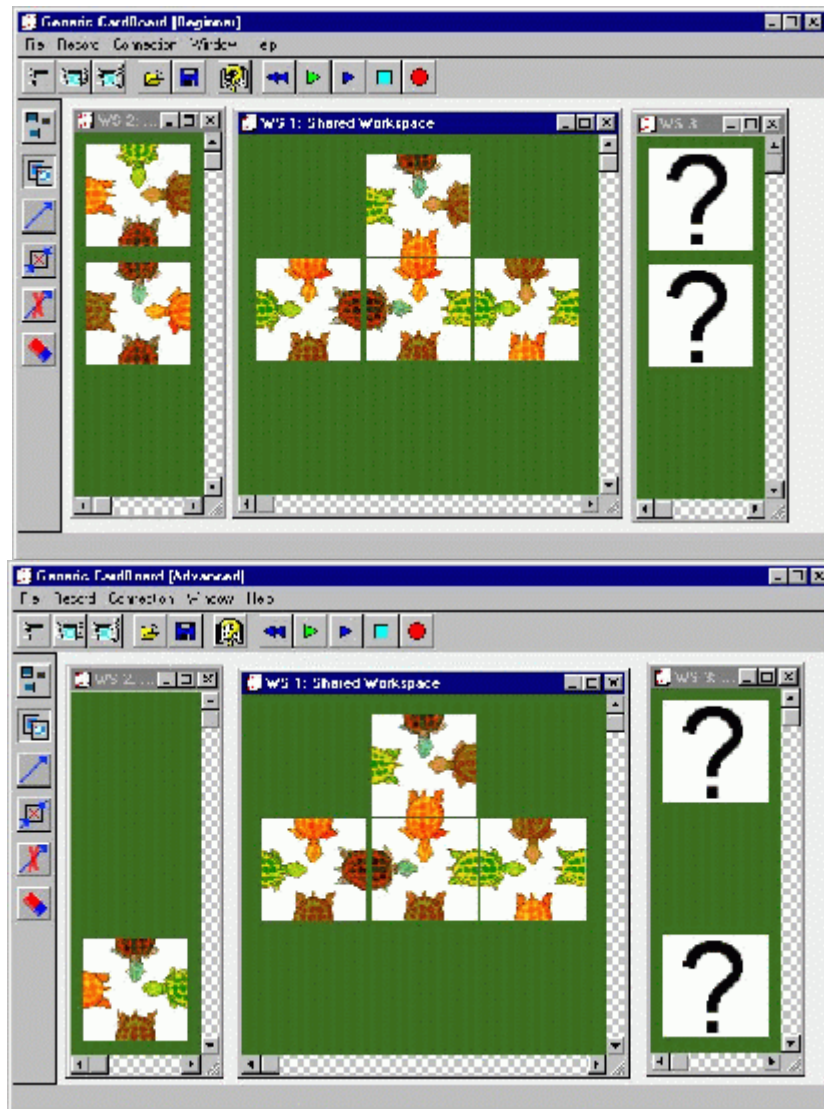


Figure 9: User interfaces for the turtle puzzle

USING THE SYSTEM

Generally, the ODLE framework system is designed for diverse settings, such as settings for primary schools and for continuing education, and for business settings, etc. However, a specific application of the framework system has to take the needs of a certain setting into account. In this section, we will exemplify a scenario of how the framework could be applied in a course of high school mechanics. This example is based on existing system components. During the sequence of interactions with the system, different features of the framework system will be used.

Figure 10 shows a snapshot of some workspaces that developed from a given exercise (1) to the derivation tree created by a student (2). The *Query Result* workspace (3) presents the automatically generated feedback, and the fourth workspace (4) contains a dialogue with a human tutor.

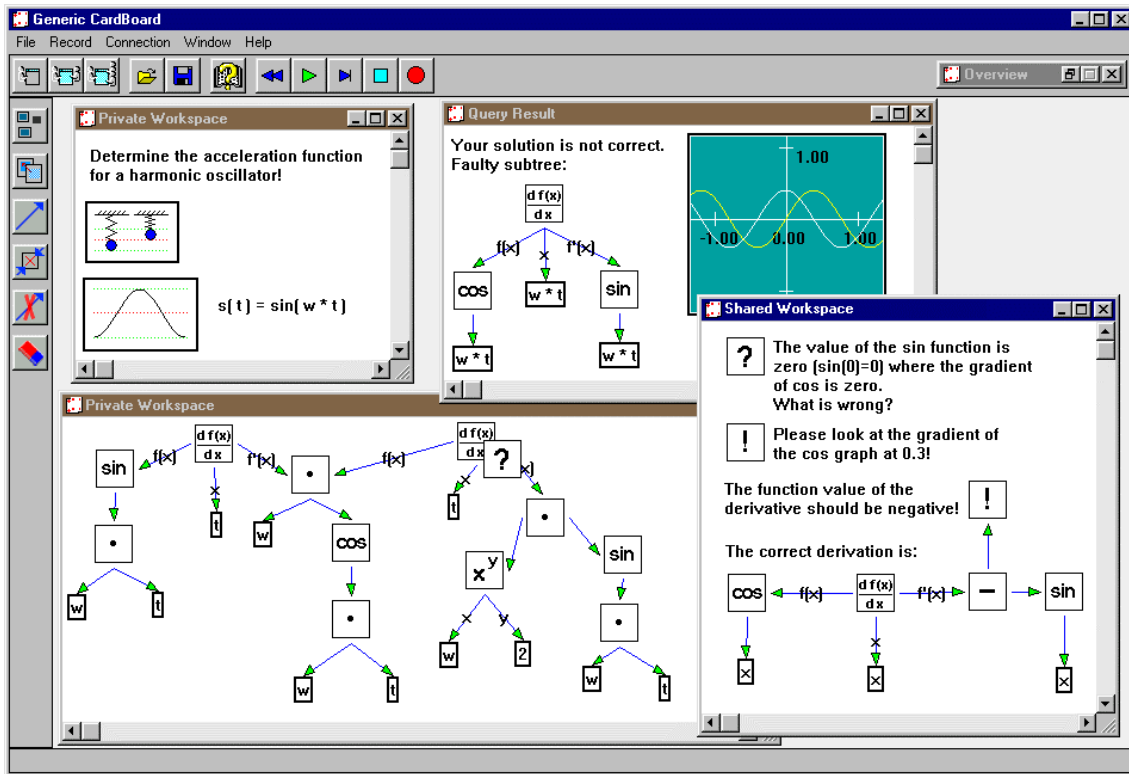


Figure 10: Scenario of a task solution in high school mechanics

1. Presentation of the task (workspace at the top left in figure 10)

The task for the student here is to determine the acceleration function of a harmonic oscillator. The s - t -relation of the movement of a harmonic oscillator (an ideal spring) is given.³ The content of this workspace initially has been loaded from a workspace repository of tasks in mechanics. The student has to calculate the a - t -relation, that is the time-dependent acceleration of the small ball attached to the end of the spring.⁴

2. Constructing derivatives (workspace at the bottom left)

The crucial point in solving the task is to build the first and second derivative of the function $s(t)$. The general idea is to support the derivation task by a card net representation of the term structure. This representation helps to illustrate the operations involved and to reference the source of possible mistakes.

The tree representation of the initial function $\sin(w \cdot t)$ would be generated from a textual representation by the system on demand. The student then constructs the derivatives by drag and drop manipulations on cards. Eventually, dropping a question card on one of the derivation symbols (see figure 10) would initiate the analysis of the respective part of the card net by the system.

3. Solution analysis (workspace *Query Result* at the top right)

³ w stands for ω . The argument of sin is $2 \cdot \pi \cdot f \cdot t$.

⁴ A correct solution plan would be:

$$s(t) = \sin(\omega \cdot t); v(t) = \frac{d}{dt} s(t) = \omega \cdot \cos(\omega \cdot t); a(t) = \frac{d^2}{dt^2} s(t) = \omega^2 \cdot (-\sin(\omega \cdot t)).$$

The analysis is carried out by the same student modelling mechanism as in the *SuperDeriv* application (cf. section 5). The component has been reused here by “plugging” it into a different user interface. The diagnosing agent finds a possible mistake in a sub-derivative.⁵ For this reason, the agent generates the *Query Result* workspace and illustrates the mistake by giving a partial tree in the card representation.

Alternatively, when using an exclamation card instead of the question card, the agent would manipulate the student's card net to correct the mistake.

4. Getting further help (*Shared Workspace* at the bottom right)

In this *Shared Workspace* the student discusses his/her mistake with a human tutor by using arithmetic and request cards. This is done by dragging existing cards from other workspaces (including the *Query Result* workspace) to the shared workspace.

The tutor has access to the recent flow of information between the student and the support agent and has therefore the knowledge about the specific problem. In this example, the student argues that his solution fulfills the premise that a derivative has to have a zero value, where the function has an extreme value.⁶

The tutor now proposes the student to look at the gradient of the sin function at 0.3. The function falls, the gradient is negative and therefore the derivative must be negative, too. The tutor finally points out the minus sign of the sin function to the student and provides the correct sub-solution.

The four workspaces in this example differ in terms of the pre-defined card sets and in the system support that is given. These workspace properties are summarized in table 3.

Table 3: Properties of workspaces in mechanics example

Workspace	Initiated	Content created	Synchronized	Card set	System support
1 (top left)	by user	from file	no	text, image and arithmetic cards	none
2 (bottom left)	by user	by user and agent	no	Arithmetic and query cards	solution analysis, correcting solution
3 (top right)	by agent	by agent	no	text, image and arithmetic cards	illustration of possible mistake
4 (bottom right)	by user	by user and tutor	yes	text, arithmetic and query cards	none

⁵ It is a common mistake to derive *sin* from

$$\cos\left(\frac{d}{dx} \cos(x)\right) \neq \sin(x). \text{ The correct derivative would be } (-\sin).$$

⁶ $\frac{d}{dt} f(s) = 0 \equiv f$ has a local extremum at x .

CONCLUSION

Our experience with the described approach is very encouraging. The new architecture makes the distributed interactive learning environment completely independent of the monitoring and modeling system. Yet, intelligent support can easily be added by connecting the DALIS server to the MATCHMAKER server, given the availability of adequate definitions for monitoring and modeling agents as separate knowledge bases. From an engineering point of view, the provision of individual and group support is facilitated as compared to an implementation “from scratch”, since the architecture provides a high level interface for the flow of information between the interactive applications and the modeling system both in terms of data formats and flexible control mechanisms.

The period of work reflected in this paper was mainly dedicated to implementing a new flexible and powerful framework system as a platform for the development of intelligently supported ODLE. This work was inspired by a rich body of previous experience in both modeling and the construction of interactive, distributed educational applications. Moreover, the usability of the framework system has been indicated by a number of sample applications as presented in this paper. For the future, we expect innovative types of CSCL applications, mainly characterized by the flexible “symbiotic” interaction between human and artificial agents to emerge from this framework.

Originally, it has been foreseen to port the C⁺⁺ version of the MATCHMAKER toolbox from the Win32 platform to other platforms (especially to X/OSF Motif). With the appearance of Java, a programming language that provides robust object oriented programming based on a rich set of standard classes, system independent user interface design and high level inter-process communication handling (RMI, CORBA), this plan has been changed in favor of an enhanced rebuild of the MATCHMAKER toolbox in Java (JMM). JMM will provide coupling facilities for *existing* Java applications and a very dynamic way of evaluating remote *function* calls in distributed processes (RMI, REFLECTION). First prototypical applications have already been implemented on the basis of JMM (which is currently in a beta-status).

The further development of the agent framework is characterized by two opposing forces, i.e. the need for concurrent and distributed processing versus the goal of integration and abstraction of user data and the handling of system complexity. It seems that the tradeoff between those positions depends, to a certain degree, on the specific applications. The domain-specific applications mentioned in this paper have been build on the basis of the distributed DALIS framework, whereas the domain-independent framework on the basis of *CardDalis* is currently established as a single-agent system.

Ongoing work is dedicated to the provision of mainly domain-independent components for the monitoring and support of group interactions. Of course, the DALIS agent system can be simplified in such a way as to ignoring the “authorship” of individual contributions in a group setting with synchronized user interfaces. In this indiscriminate view, the whole group is treated as a single user. The same mechanisms that monitor the behavior of individual users can then be used to evaluate the success of the group formation. Closing the “open loop” in such a way, the same feedback as for single users will also be provided by the system for groups.

However, the situation gets much more complicated when differentiating between individual contributions in a shared, i.e., synchronized workspace, since it is hard to determine by the system if all users would agree on individual contributions and if all group members have a share understanding of the issues involved. Concerning group supervision and support, this goal is expected to be achieved in rather problem solving oriented tasks. Furthermore, remediation in group settings has to be based on a well-balanced initiative between the system, the users and the tutor.

References

- Cheikes, B. A. 1994, GIA: An agent based architecture for intelligent tutoring system, In *Proceedings of the Third International Conference on Information and Knowledge Management*, Galthersburg, Maryland.

- Dillenbourg, P., Baker, M., Blaye, A. O'Malley, C. 1995, The evolution of research on collaborative learning, In P. Reimann H. Spada (eds.), *Learning in Humans and Machines: Towards an Interdisciplinary Learning Science* (189-211), Oxford: Elsevier.
- Finin, T., Fritzson, R., McKay, D. McEntire, R. 1994, KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, Galthersburg, Maryland.
- Finin, T., Labrou, Y. Mayfield, J. 1997, KQML as an agent communication language, In J. M. Bradshaw (Ed), *Software Agents* (291-316), Menlo Park, California: AAAI Press/MIT Press.
- Gassner, K., Tewissen, F., Mühlenbrock, M., Loesch, A. Hoppe, H. U. 1998, Intelligently supported collaborative learning environments based on visual languages: A generic approach, In *Proceedings of the Third International Conference on the Design of Cooperative Systems COOP-98*, Cannes, France.
- Genesereth, M. R. 1997, An agent-based framework for interoperability, In J. M. Bradshaw (Ed), *Software Agents* (317-345), Menlo Park, California: AAAI Press/MIT Press.
- Genesereth, M. R., Singh, N. P. Syed, M. A. 1994, A distributed and anonymous knowledge sharing approach to software interoperation, In *Proceedings of the CIKM Workshop on Intelligent Information Agents*, Galthersburg, Maryland.
- Hoppe, H. U. 1994, Deductive error diagnosis and inductive error generalization for intelligent tutoring systems, *Journal of AI in Education*, 5(1), 27-49.
- Hoppe, H. U. 1995, The use of multiple student modeling to parameterize group learning, In J. Greer (Ed), *Proceedings of AI-ED 95*, Washington D.C., USA.
- Hoppe, H. U., Baloian, N., Schupp, M. Zhao, J. 1995, Eine Architektur für die Unterstützung von unmoderierten Gruppen-Lernprozessen (An architecture for supporting unmoderated group learning processes), In E. Schoop, R. Witt U. Glowalla (Eds), *Hypermedia in der Aus- und Weiterbildung* (55-64), Universitätsverlag Konstanz.
- McCalla, G. I., Greer, J. E., Kumar, V. S., Meagher, P., Collins, J. A., Tkatch, R. Parkinson, B. 1997, A peer help system for workplace training, In B. du Boulay R. Mizoguchi (Eds), *Artificial Intelligence in Education: Knowledge and Media in Learning Systems* (183-190), Kobe, Japan: IOS Press.
- Mühlenbrock, M., Tewissen, F. Hoppe, H. U. 1997, A framework system for intelligent support in open distributed learning environments, In B. du Boulay R. Mizoguchi (Eds), *Artificial Intelligence in Education: Knowledge and Media in Learning Systems* (191-198), Kobe, Japan: IOS Press.
- Paiva, A. 1996, Learner modelling agents, In P. Brna, A. Paiva J. Self (Eds), *Proceedings of the European Conference on Artificial Intelligence in Education*, Lisbon, Portugal.
- Paiva, A. 1997, Learner modeling for collaborative learning environments, In B. du Boulay R. Mizoguchi (Eds), *Artificial Intelligence in Education: Knowledge and Media in Learning Systems* (215-222), Kobe, Japan: IOS Press.
- Plötzner, R., Hoppe, H. U., Fehse, E., Nolte, C. Tewissen, F. 1996, Model-based design of activity spaces for collaborative problem solving and learning, In P. Brna, A. Paiva J. Self (Eds), *Proceedings of the European Conference on Artificial Intelligence in Education*, Lisbon, Portugal.
- Ritter, S. Koedinger, K. R. 1996, An architecture for plug-in tutor agents, *Journal of Artificial Intelligence in Education*, 7(3/4), 315-347.
- Self, J. 1994, The role of student models in learning environments, *IEICE Transactions on Information & Systems*, E77-D(1), 3-8.
- Suthers, D. 1996, Architectures and methods for designing cost-effective and reusable ITSs, (<http://advlearn.lrdc.pitt.edu/its-arch/>)
- Tewissen, F. 1996, Begriffsnetze als Basis für ein System zur kooperativen Lösung physikalischer Aufgabenstellungen (Concept maps as a basis for a system supporting the cooperative solution of physics problems). University of Duisburg. (in German)
- Zhao, J. Hoppe, H. U. 1994, Supporting flexible communication in heterogeneous multi-user environments, In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems* (442-449), Juan-les-Pins, France.