Educational Services in the ActiveMath Learning Environment

Carsten Ullrich [a]
[a] *Shanghai Jiaotong University, Shanghai, China*

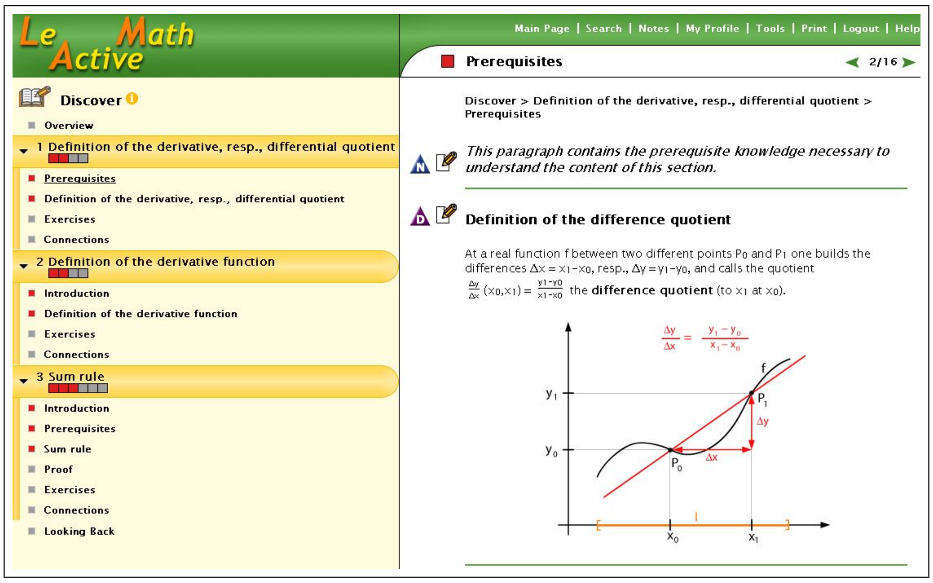
Paul Libbrecht [b]
[b] *DFKI GmbH, Saarbrücken, Germany, http://www.activemath.org/*

Abstract. ActiveMath is a Web-based learning environment for mathematics. In this article, we discuss ActiveMath from a service and semantics perspective. We describe the service-oriented technologies of ActiveMath and how the services interplay with the domain semantics (mathematics) and the pedagogical semantics used, e. g., for course generation. More specifically, we provide details on the knowledge representation for mathematics we use in ActiveMath and how it is used to represent learning materials. We also elaborate on the representation of learning objects from a pedagogical point-of-view, that is, how we capture the instructional semantics. We then show how learning goals are represented and how a course generator assembles sequences of learning objects to fulfill these goals. Some tools in ActiveMath are client-based and require different techniques. We use our assembly tool as an example to illustrate how ActiveMath integrates client-based tools.

Keywords. service-oriented-architecture, web-based, learning-environment, semantics, education

INTRODUCTION

Service oriented architectures are becoming increasingly popular [19]. Their basic paradigm is convincingly straightforward: encapsulate specific functionality that is of common interest into a single service which is available to third-parties. In this chapter, we will discuss the service architecture of ActiveMath, a Web-based learning environment. In the FP6 project Le ActiveMath, we used the service paradigm to tackle a number of issues arising in distributed or grid-based learning environments. Service-orientation in ActiveMath takes place on different levels and involves different technology, ranging from XML-RPC to WSDL. In the following, we will first introduce ActiveMath (Section 1) and basic service-techniques used in ActiveMath (Section 2). Then, we will describe the different services and motivate our decisions for the specific protocols. We will provide details on the different semantics used in ActiveMath: the mathematical and educational semantics and show how the main components operate on these semantics (Section 3). We close this chapter by a discussing of related work (Section 4) and



Le Active Math
Main Page | Search | Notes | My Profile | Tools | Print | Logout | Help
Prerequisites
2/16
Discover
Overview
1 Definition of the derivative, resp., differential quotient
Prerequisites
Definition of the derivative, resp., differential quotient
Exercises
Connections
2 Definition of the derivative function
Introduction
Definition of the derivative function
Exercises
Connections
3 Sum rule
Introduction
Prerequisites
Sum rule
Proof
Exercises
Connections
Looking Back
Discover > Definition of the derivative, resp., differential quotient > Prerequisites

This paragraph contains the prerequisite knowledge necessary to understand the content of this section.

Definition of the difference quotient

At a real function f between two different points P_0 and P_1 one builds the differences $\Delta x = x_1 - x_0$, resp., $\Delta y = y_1 - y_0$, and calls the quotient $\frac{\Delta y}{\Delta x}(x_0, x_1) = \frac{y_1 - y_0}{x_1 - x_0}$ the **difference quotient** (to x_1 at x_0).

$\frac{\Delta y}{\Delta x} = \frac{y_1 - y_0}{x_1 - x_0}$
f
y_1
y_0
P_1
P_0
Δy
Δx
I
x_0
x_1

Figure 1. A course presented in ActiveMath

the lessons we learned from the application of services in the Le ActiveMath project.

1. THE LEARNING ENVIRONMENT ACTIVEMATH

ActiveMath [21,23] is a Web-based intelligent learning environment for mathematics that has been developed since the year 2000 at the Saarland University and at the German Research Center of Artificial Intelligence (DFKI).[1]

ActiveMath uses an extension of OMDoc [16,22] to encode its educational resources. In addition to presenting pre-defined interactive materials, it uses a course generator (called Paigos) for the dynamic and adaptive assembly of structured sequences of learning objects. Figure 1 contains a screenshot of a course presented in ActiveMath.

A presentation component transforms the OMDoc documents represented in XML to the desired output format, e. g., HTML, XHTML +MathML, and PDF. A learner model stores the learning history, the user's profile and preferences, and a set of beliefs that the systems holds about the cognitive and meta-cognitive competencies and the motivational state of the learner. The domain model that underlies the structure of the learner model is inferred from the content for that domain and its metadata.

A complex subsystem in its own right is ActiveMath's exercise subsystem [9] that plays interactive exercises, computes diagnoses and provides feedback to the learner in a highly personalized way. It reports events to inform the other components about the users' actions.

[1] `http://www.activemath.org/`.

In 2007, at the time of this writing, a significant amount of educational resources exists in ActiveMath's repositories for Fractions (German), Differential Calculus (German, English, Spanish) at high school and first year university level, Operations Research (Russian, English), Methods of Optimization (Russian), Statistics and Probability Calculus (German), Matheführerschein (German), and a Calculus course from University of Westminster in London.

To realize a smooth and efficient cooperation of all components and in order to integrate further internal and external services, ActiveMath has a modular service-oriented architecture. It includes the xml-rpc Web communication protocol for simplicity and remote support. In addition, an event framework enables asynchronous messaging between system components. Communication with third-parties components is done by Web-services using WSDL and SOAP. We will now discuss these service technologies.

2. SERVICE TECHNOLOGIES USED IN ACTIVEMATH

In this section we first survey the basic needs of a Web-service architecture for ActiveMath then focus on the technologies we used.

The internal communication within the server is governed by pragmatic requirements, mostly targeting at an efficient communication between different components. ActiveMath components can be distributed, e. g., the domain knowledge base can run on a different machine than the course generator. On this level, communication happens via direct Java calls or xml-rpc for efficiency reasons (see the lessons learned, in the final section).

If one leaves the restricted view of a single server-based environment, the picture changes. For instance, the generic integration of third-party repositories is much easier to achieve if standard Web-service techniques are employed. Similarly, making functionality such as course generation available to other learning environment makes more sense if the interfaces follow standards, otherwise they will rarely be used.

It is thus of great importance to provide standards-based methods, using communication interfaces that expose their documentations both to for other machines to use or for humans to use. The usage of SOAP , WSDL, XML-schema, and re-use of widespread schema parts forms an important requirement for a greater acceptance.

For the interfaces to the client used by the learner, beyond the simple browser-content delivery, a less principled approach was chosen because each client tool was customized for the ActiveMath usage and because we tried to keep the requirements on the client side fairly low, in terms of available software components and resource. Our requirement for rich-client tools were the following:

- full contextualization resulting in the start of a tool at a single click.
- privacy enforcement: individual tools launched on the client may have the ability to notify the server following an action of the user; more generally, the tools may read or write information that pertain to learner's activities: by means of an appropriate contextualization, they will have channels to write or read information about the user; these channels should only provide

access to the user's own information so as to prevent even maliciously manipulated client tools to effect other user's activity.
- usage of the same communication channel as the browser since in several cases, security concerns on corporate networks allow only basic forms of HTTP communication (generally served through a proxy).
- consistent user-experience: at a higher level, the integrated platform should be felt as such. For instance, tools that allow input of mathematical-formulæ or content-item references can receive pasted content copied from the presentation. Similarly, the language should be integrated and reference to particular learning concepts should be linked to it as much as possible.

As a consequence of these various requirements, ActiveMath uses several service oriented technologies, which we will describe in the following.

2.1. XMLRPC

Several of ActiveMath services are based on the XML-RPC communication protocol because of its web-based nature, its simplicity, and very broad platform support. XML-RPC is a simple HTTP POST based encoding of remote-procedure call, similar to SOAP's RPC profile but with a much stricter encoding format which has, as consequence, for example, that XML content needs an extra packaging (for example be delivered as strings).

XML-RPC was chosen early in the project which has allowed us to start a distributed architecture quickly. The performance of XML-RPC was always quite good. Since then, further service-technologies have been investigated as can be seen below but all basic remote-procedure-calls still use this standard.

2.2. EVENT MODEL

For many situations and applications however, a remote procedure call (ie, synchronous communication) is inappropriate. In particular, some components need just a quick notification when events of interest occur in another service or component. Moreover, the sending component may not be aware of all services which need to be informed.

The components interested in such messages may not be known in advance or can change over time or depending on configuration. To simplify a flexible integration and reuse of Web-services, a mechanism for registering interest and for propagating events is needed. Therefore, the XML-RPC approach is complemented by an event framework in ActiveMath now.

Events are a mechanism for a powerful and flexible, yet rather loose integration of components. The event paradigm is especially useful for state update notifications and reactions to them. Similar messaging and notification frameworks exist for SOAP. Since we use XML-RPC rather than SOAP for simplicity and performance reasons, we devised a new lightweight event framework.

An example for event "publication" is the following: when the learner finished working on an exercise, the exercise subsystem issues an event. The event carries information describing the learner, the identifier of the exercise, the success rate, a diagnosis about the learner's mastery, the time stamp of the event, etc. Listeners to such an event can be the learner model as well as the suggestor of the tutorial component.

A component that publishes events is an event source. A component that subscribes to the events published by an event source is called a listener which receives event messages from the event source.

In contrast to a full-fledged messaging model, events are not sent from a specific sender to a specific recipient but remain anonymous: when publishing an event, the component is usually not aware who is listening to the events (only the module managing the subscriptions is). Also, usually the listener does not care which component or module created the event, it only knows where to subscribe to the events it is interested in.

Each event message object consists of attributes. The attributes common to all ActiveMath events are:

- a type, indicating what happened (eg, "user X has logged in", "new content available"). Depending on the type, the event may carry additional data for further describing the event.
- a timestamp, indicating when it happened: Events happen at a single point in time (and space). The timestamp indicates the wall-clock time of the event source when the event took place.
- a source, indicating where happened. This is typically the name of the producer, ie, the class name of the component that produced the event.

Other attributes are specific to the type of events.

It is often desirable to group event types across several dimensions, eg, for filtering ("I want all event concerning a user", "Give me all events related to the Dictionary"). However, events don't fit well into a rigid type hierarchy: For instance, consider the even "user logged in". What would be the "natural" group of such an event? An application event? An user event? An interaction event? What's needed is a flexible, non-hierarchical type framework that allows for a mixing of characteristics such as "is associated to a user". Therefore, ActiveMath uses event tags. Tags are labels for event types. Each event type can be associated to zero or more tags. Tag definitions can be nested, that is, a tag can inherit from another one. Multiple inheritance is allowed.

Tags can add attributes to an event. For example, the "user" event tag adds the attribute userId to an event. Tags defined without attributes (such as "application") just serve as marker tags. The following attributes are found in event tags, among others: user event tag (identifier of the user that caused this event, empty if user is anonymous); session event tag; item event tag (the identifier of the item, interaction and application event tag.

Remote eventing based on XML-RPC has been implemented for both server-to-server and rich-client-to-server communication. A special case is the event exchange with the browser client. Our solution is based on asynchronous client-server communication using the browser's XMLHttpRequest object. For XML-RPC, the attribute values are simple XML-RPC datatypes. Therefore, attribute names or an event type must be unique across the event type's class, the event tags, and the event base class.

2.3. WEB-SERVICES

A Web service can be defined as follows [3]: “a a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.”

Standards for each of the given properties exists: WSDL describes a service independent from its underlying implementation by specifying the interfaces and their bindings. SOAP defines the structure and types of messages that are exchanged between server and client. Service discovery is supported by UDDI.

We will now describe educational services in ActiveMath that use the above technologies.

3. SEMANTIC SERVICES IN ACTIVEMATH

ActiveMath uses semantics to represent the domain knowledge, that is the subject domain, namely mathematics, and to represent the pedagogical knowledge involved in course generation (the assembling of learning objects that make up the domain knowledge to form courses). Different services operate on these semantics.

We will first discuss semantics and services from the domain point of view (mathematics): the knowledge base (Section 3.1), followed by the representation of the domain knowledge (Section 3.2). Then, we concentrate on the use from the pedagogical point of view: how to represent the pedagogical purpose of a learning object (Section 3.3) and how to use this semantic to make content from different repositories accessible to a learning environment (Section 3.4). The subsequent sections investigate learning goals, namely how to represent them (Section 3.5) and how to generate sequences of learning objects that fulfill them (Section 3.6). Finally, we discuss a client-rich tool that exemplifies how to client applications can use the above semantics (Section 3.7).

3.1. KNOWLEDGE BASE SERVICES

ActiveMath clearly separates the content from the functionalities working on it (such as course generation and presentation). Our knowledge base serves as a generic storage for the OMDoc-documents in ActiveMath. Queries include the retrieval of xml-content/items based on identifiers and search based on relations between items in both directions. Currently, fuzzy and semantic search are supported by this database.

At load time, the knowledge base resolves references relative to the mathematical theory and collection the element is in, into absolute references that include the theory and collection explicitly. This makes the structuring more flexible, integration more robust, and eases modifications. For communication, the knowledge base service provides an xml-rpc protocol. This was chosen to balance expressivity and performance.

One objective of using a generic semantic markup language is to keep the encoded content reusable and interoperable by other, even non-educational, mathematical applications. ActiveMath uses the semantic xml-markup language for mathematical documents, OMDoc [16,22], for its content encoding. OMDoc has evolved as an extension of the OpenMath (European) standard for mathematical symbols and expressions [6]. ActiveMath uses these representations for the following reasons: they provide a semantics for mathematical expressions and a standardized structure of the content. Especially OpenMath is used for a variety of mathematical services, from mathematical evaluation until presentation, and thus, all applications can take advantage of each other's advances in tools and implementations.

ActiveMath's content is represented by a collection of typed items called "learning objects" annotated with metadata. The semantic information includes types, relations, and other mathematical and educational metadata. The type indicates a characterization of the items as collection, theory, concept or satellite items: an OpenMath symbol defines a mathematical concept abstractly; a theory assembles concepts and it can import other theories; concepts (definitions, algorithms, and assertions/theorems) are the main items of mathematical contents, whereas satellites (exercises, examples, explanations, introductions) are additional items of the content which are related to one or several concepts. All items are accessible via a unique identifier. The content (including the OpenMath symbols) induces an ontology.

For ActiveMath, we have extended the primarily mathematical OMDoc to serve educational purposes. For this, the metadata characterize not only mathematical, organizational, and intellectual property rights properties such as Dublin Core, but also educational annotations. These lom compatible metadata [36] include properties such as difficulty, learning context, and field.

Moreover, items can be linked by relations. The `domain_prerequisite` relation expresses a mathematical dependency between concepts, ie, it describes which concepts are mathematically necessary in order to define the current one. The `for` relation links satellite items to concepts and definitions to symbols. For instance, an example or exercise may illustrate or train a related definition. The `against` relation describes an item as a counter example or misconception of the related concept. The `is_a` relation serves to represent mathematical hierarchies and special cases, eg, a "function" is a specific "relation".

Note that several (mathematically equivalent) definitions may exist for one symbol, eg, the convergence of a function can be defined via $\epsilon - \delta$ fomulae or via sequences. Definitions and theorems can be different for different learning-contexts or fields too. That is, only the OpenMath/symbol layer of the induced ontology is abstract in the sense that it represents concepts uniquely as required in certain ITS applications. ActiveMath items are multilingual but the system does not yet handle different versions of one and the same element which have the same metadata (e. g., learning context).

As we shall see in the following, the types and metadata properties and relations of the learning objects can be exploited to perform very advanced dynamic adaptation of the learning material, sequencing and presentation to the learner's needs and the context. Why is the semantic representation of symbols, ie, OpenMath, then needed in addition?

1. In ActiveMath the evaluation of the learner's input for questions and exercises can "subcontract" Computer Algebra Services, CAS, (possibly different CAS, whichever is suitable). For this, any of the CAS service has to "understand" the meaning of mathematical expressions, which is realized via the standardized semantic representation OpenMath. The language of a specific CAS or presentation MathML is not sufficient for that purpose.
2. Based on the OpenMath semantic representation, ActiveMath' presentation component can render mathematical formulæ appropriately for the needs and context of the learner, e. g., country-dependent notation and various output formats (html, xhtml +MathML, pdf). This is a feature unique to ActiveMath currently.
3. In ActiveMath, copy& paste of formulæ is possible based on references to formulæ semantics hidden in the presentation. This is a feature which is unique for ActiveMath. More details about usages of this encoding in input and output of mathematical formulæ is provided in the section 3.7.

3.3. AN ONTOLOGY OF INSTRUCTIONAL OBJECTS

A first step towards intelligent services is to define the terms used in the application domain [28]. For educational services the terms need to describe the educational resources (or learning objects) as well as the learning goals. We designed an ontology of instructional objects (oio) that was developed to characterize educational resources [37]. Although originally developed for mathematical resources, it can also be used for describing other subject domains, as long as the domain can be structured in distinct elements with relations (e. g., physics). The oio describes resources sufficiently precise for a pedagogically complex functionality such as course generation.

Seminal work on using ontologies for e-learning was done in the ISIR lab, headed by Mizoguchi: they sketched out how ontologies can help to overcome problems in artificial intelligence in education [28]; and described how an assistant layer uses an ontology to support the complete authoring process, for instance by giving hints on the course structure [2,11].

The oio has a more specific scope; instead of describing the authoring process during which the educational resources are developed, the ontology is focused on describing the resources. It thus defines a set of types (or classes) that is used to annotate educational resources.

The ontology is described in detail elsewhere [37]. Central to the ontology (shown in Figure 2) is the distinction between the classes `fundamental` and `auxiliary`. The class `fundamental` subsumes instructional objects that describe the central pieces of domain knowledge (concepts). Auxiliary elements include instructional objects which contain additional information about the fundamentals as well as training and learning experience.

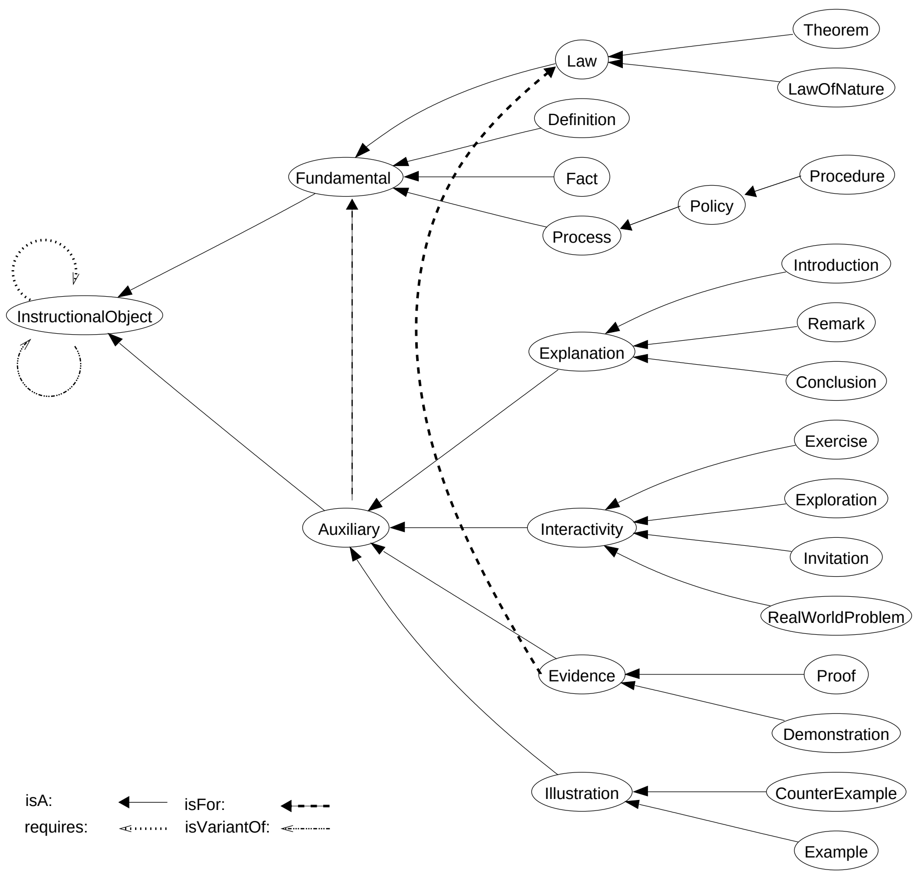


Figure 2. Overview of the Ontology of Instructional Objects

3.4. A MEDIATOR FOR REPOSITORY INTEGRATION

The ontology described in the previous section allows specifying the instructional function of educational resources and contains necessary information required for automatic, intelligent course generation. However, as a matter of fact, most of the repositories available today use their own metadata schema rather than the terms defined in the OIO. Yet, the pedagogical knowledge formalized in the course generator should be independent of the concrete metadata used in the repositories, as one wants to avoid designing separate knowledge for each repository.

The challenge of providing uniform access to resources has been recognized since long. Mediation information systems are a well-known solution to this challenge [41]. Its main component, called *mediator*, offers a uniform interface for accessing multiple heterogeneous data stores (e. g., file systems, different databases, . . .). The mediator uses mappings between the data representation used in the mediator and those used in the repository to translate queries. Each repository is enhanced with a wrapper, which can be integrated into the repository itself or into the mediator. This way, the query component does not have to know the specification of the data sources and their query languages.

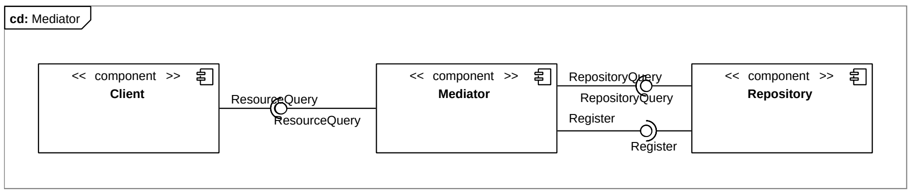


Figure 3. Overview of the mediator

Any mediator used for course generation needs to answer queries extremely quickly: as our evaluations have shown generating an expanded course for a single concept results in about 1 500 mediator queries that are expanded to more than 11 000 queries to the repository.

Existing ontology-based query rewriting approaches [8] are very expressive, but, in consequence, complicated to use and not optimized for efficiency. Therefore, we decided to develop a query rewriting approach which is less powerful than other systems but expressive enough for our translation purposes. This specialization allows for optimizations, e. g., during query processing.

Additionally, in the developed framework it is easy to integrate new repositories, by specifying a mapping between the OIO and the representation used in the repositories and implementing a small set of interfaces.

The mediator described in this section answers queries about the existence of specific educational resources. It is illustrated in Figure 3. Its interfaces[2] are Web-service interfaces. A repository can register (interface `Register`), passing an OWL representation of its metadata structure as well as a mapping of the OIO onto this representation to the mediator. Additionally, each repository needs to implement the interface `RepositoryQuery` that allows the mediator to query the existence of educational resources. Clients access the mediator using the interface `ResourceQuery`.

3.4.1. Querying the Mediator

The interface `ResourceQuery` takes a partial metadata description as input and returns the set of identifiers of the educational resources that meet the description. The metadata used in a query sent to the mediator must comply to the OIO. It can consist of three parts:

- Class queries specify the classes the educational resources have to belong to. They consist of a set of tuples (`class` *c*) in which *c* denotes the class of the OIO the returned educational resources must belong to.
- Property queries specify property metadata. They consist of a set of triples (`property` *prop val*), where *prop* and *val* are property and value names from the OIO. Retrieved educational resources have to satisfy each given property-value pair.
- Relation queries specify the relational metadata the educational resources have to meet. They consist of a set of triples (`relation` *rel id*) in which

[2]The figure uses UML ball-and-socket icons. A ball represents a provided interface, a socket a required interface.

rel specifies the relation that must hold between the resource referenced by the identifier *id* and the educational resources to be retrieved. *rel* is a relation name of the OIO.

A query asking for all resources with an easy difficulty level illustrating the definition `def_slope` looks as follows:

```
(relation isFor def_slope) (class illustration)
(property hasDifficulty easy)
```

While processing a query, the mediator uses the information of the OIO to expand a query to subclasses. Hence, if asked for class c, the mediator returns resources belonging to c and to its subclasses.

3.4.2. Ontology Mapping and Query Rewriting

A repository that registers itself using the interface `Register` must provide sufficient information to enable the mediator to translate the metadata of an incoming query to the metadata used by the repository.

The translation is performed by an ontology-based query-rewriting mechanism. The mechanism requires an ontological representation O of the metadata structure used by the repository and an ontology mapping M. It uses O and M to compute the rewriting steps for translating the queries it receives. A registration time, a repository passes both the ontology and the mapping to the mediator. The mappings currently used by the mediator were produced beforehand by the developers.

We designed an XML-based ontology mapping language that represents the mappings between the OIO and the target ontologies.

An ontology mapping contains a set of mapping patterns, where each mapping pattern m consists of a matching pattern mp and a set of replacement patterns $RP = \{rp_1, \ldots, rp_n\}$. mp and RP consist of terms of the OIO and the ontology of the repository, respectively. A mapping pattern $m_1 = (mp_1, RP_1)$ is more specific than a pattern $m_2 = (mp_2, RP_2)$ if $mp_2 \subset mp_1$.

The idea of the ontology mapping is that every part of a query that matches a matching pattern is replaced by the replacement patterns. More formally, we say a mapping pattern $m = (mp, \{rp_1, \ldots, rp_n\})$ matches a query q if q contains each term specified in mp. Applying m to q results in new queries $q_1, \ldots, q_n$, which are derived by replacing each term of mp by the terms of $rp_1, \ldots, rp_n$, respectively.

The ontology mapping procedure applies the most specific mapping pattern to a query q. Currently, the author of an ontology mapping has to ensure manually that there is only one such pattern; future work will investigate how to support this automatically. A term for which no matching pattern is found is left as it is. This approach avoids writing mapping patterns that express the identity of terms (the use cases have shown that this is the most frequently occurring case).

3.4.3. Repository interface and caching

In order to be accessible from the mediator, a repository must implement the following interface that provides information about each of the above query types:

- `public Set queryClass(String id)` returns the classes a given resource belongs to.
- `public Set queryRelation(String rel, String id)` returns the set of identifiers of those educational resources the resource `id` is related to via the relation `rel`.
- `public Set queryProperty(String id)` returns the set of property-value pairs the given resource has.

In real use, performance matters and query processing is often time consuming mostly because of latency of the Web. For instance, course generation involves a significant amount of queries (about 100 000 expanded queries for a course for 20 concepts). Therefore, the amount of processed queries has to be reduced. We tackled this problem by integrating a caching mechanism into the mediator. If the same query (or sub-query) is sent repeatedly, the mediator does not query each connected repository again. Instead, it returns the cached set of identifiers, which increases run-time performance dramatically. The results of partial queries are cached, too.

Please note that our approach focuses on mapping of pedagogical concepts and not on mapping of instances of the subject domain. Thus, the mediator cannot yet use the information that fundamental c_1 in repository r_1 represents the same domain entity as fundamental c_2 in repository r_2, say `def_group` in r_1 and `definition_gruppe` in r_2 both define the same mathematical concept *group*. In future work, we will integrate an instance mapping technology that maps domain ontologies.

3.4.4. Limitations of the Mediator as an Educational Service

The mediator allows access to resources based on their instructional function: a service generates partial metadata, and the mediator retrieves a list of corresponding educational resources. However, this is a basic service, which has some limitations:

- How to come up with the metadata? Determining the appropriate metadata for finding educational resources is not trivial. Assume a service wants to present an example of "average slope" to the learner. Is the learning context relevant? Should it be an easy or a difficult example? These decisions depend on the current learning goal, i. e., the current pedagogical task.
- Which precise educational resources to select? Typically, the mediator returns a list of resources. Which one is the most appropriate? Are they all equivalent? A too large set might indicate that the metadata was too general. However, narrowing it down might result in an over-specification an hence in an empty set.
- A single educational resource might not be sufficient to achieve learning progress. For instance, understanding content in depth requires a sequence of carefully selected educational resources. Again, the precise resources to select depend on the learning goal.

These limitations motivate the need for a course generator, i. e., a component that operationalizes the educational knowledge and provides services on a higher level of abstraction. The following sections describe the course generator.

The mediator architecture allows finding educational resources that fulfill given criteria. Typically, an agent (a learner or a machine) searches for the resources in order to achieve a learning goal. We designed an explicit and declarative representation that can be used to encode such learning goals.

A declarative representation of goals offers several advantages. First of all, it allows a system to autonomously generate actions to achieve the goal if the system uses an appropriate framework. Secondly, it provides an abstract layer that can be used for communication between systems. Instead of only being able to talk about the resources used in the learning process, systems can communicate about the purposes of the learning process. Third, it can be used to describe precisely the functionalities that the course generator offers: for each learning goal, PAIGOS can calculate a sequence of educational resources (if available) that help the learner to achieve this goal.

Existing course generators often use the domain concepts to represent learning goals. There, the generated course provides a sequence of educational resources that leads to these concepts and includes prerequisites and other resources. However, such an approach that restricts goals to resources is too limited. Depending on their current situation, learners want to achieve different objectives with the same target fundamentals, and a course should reflect the different needs associated with the objectives. For instance, a course that helps students to discover new content should differ different from a course that supports rehearsal.

Van Marcke [38] introduced the the concept of an *instructional tasks*, which helps to define learning goals in more details: an instructional task represents an activity that can be accomplished during the learning process.

Both, the content and the instructional task are essential aspects of a learning goal. Therefore, we define learning goals as a combination of the two dimensions *content* and *task*. We call instructional tasks *pedagogical objectives*, in order to distinguish them from the declarative representation of learning goals, which we call *pedagogical tasks*:

A *pedagogical task* is a tuple $t = (p, L)$, where p is an identifier of the pedagogical objective and L is a list of educational resource identifiers. L specifies the course's target fundamentals, and p influences the structure of the course and the educational resources selected. The order of the resources in L is relevant and the same task with L's elements ordered differently can result in a different course.

As an example, the educational objective to discover and understand content in depth is called `discover`. Let's assume that `def_slope` and `def_diff` are the identifiers of the educational resources that contain the definition of the mathematical fundamental "average slope of a function" and "definition of the derivative, resp., differential quotient", respectively. We can now write the learning goal of a learner who wants to discover and understand these two fundamentals as the educational task $t =$ (`discover`, (`def_slope`, `def_diff`)). The fundamentals are processed in the given order: first `def_slope`, followed by `def_diff`.

Table 1 contains a selection of pedagogical tasks formalized within ACTIVEMATH, partly designed in cooperation with pedagogical experts.

Pedagogical tasks can be "internal" tasks, used for internal course generation purposes only, or tasks that are of potential interest for other services. The

Identifier	**Description**
`discover`	Discover and understand fundamentals in depth
`rehearse`	Address weak points
`trainSet`	Increase mastery of a set of fundamentals by training
`guidedTour`	Detailed information, including prerequisites
`trainWithSingleExercice`	Increase mastery using a single exercise
`illustrate`	Improve understanding by a sequence of examples
`illustrateWithSingleExample`	Improve understanding using a single example

Table 1. A selection of pedagogical objectives used in ActiveMath

second category of tasks is called *public tasks*. Public tasks need to be described sufficiently precise in order to enable a communication between components as described above. The description designed for Paigos contains the following information:

- the identifier of the pedagogical objective;
- the number of concepts the pedagogical objective can be applied to. A task can either be applied to a single concept (cardinality 1) or multiple concepts (cardinality n).
- the type of educational resource (as defined in the oio) that the task can be applied to;
- the type of course to expect as a result. Possible values are either `course` in case a complete course is generated or `section` in case a single section is returned. Even in case the course generator selects only a single educational resource, the resource is included in a section. This is due to requirements from standards like ims cp which is used by the course generator Web-service.
- an optional element `condition` that is evaluated in order to determine whether a task can be achieved. In some situations, a service only needs to know whether a task can be achieved but not by which educational resources. In that case, the condition can be passed to the mediator, and if the return value is different from `null`, the task can be achieved.
- a concise natural language description of the purpose that is used for display in menus.

Educational tasks together with the ontology of instructional objects allow representing learning goals and the instructionally relevant aspects of resources used to achieve those goals. In the next section, I describe how the course generator Paigos uses these representations in order to assemble personalized sequences of educational resources that support the learner in achieving her learning goals.

3.6. COURSE GENERATOR

Course generation has long been a research topic [32]. It uses pedagogical knowledge to generate a structured sequence of learning objects that is adapted to the learners' compe-tencies, individual variables, and learning goals [5]. This generation happens upon request of a client (a learner or a software system). Ideally, the sequence is not a flat list of learning objects but is structured in sections and subsections. This structure can convey additional information relevant to the learning process. In course generation, the course is generated completely before it is presented to the learner. This early generation has the advantage that the course can be visualized to the learner, thereby informing her about the structure. In addition, the student can navigate freely through the course.

Early work on course generation [38] emphasized the value of the teaching knowledge used by the course generator. Today's course generators (see the section on related work) often use rather simplified pedagogical knowledge that guides the assembly, most of the time the "prerequisite"-relation that represents that understanding resource B depends on having understood resource A is used. One reason for using only very simple pedagogical knowledge is that pedagogical knowledge is hard to assess and expensive to model. Systems that aim at modeling pedagogically knowledge require a large teaching model: for instance, the instructional knowledge base of the system GTE encompasses about a hundred tasks and methods; the instructional ontology proposed in [11] consists of about 530 classes. Paigos, the course generator of ActiveMath contains about 300 rules that determine the selection, ordering and structuring of learning objects. This "expensive" functionality lends itself to being "outsourced": if the course generator is available as a service then other learning environments can access the functionality without having to re-implement it. However, none of the previously developed course generators allowed the integration of new repositories and the accessible content was restricted to the local repositories. Paigos in contrast is a Web service where clients can register their repositories and subsequently use Paigos for course generation (see Section 3.6.3).

Paigos is used to generate courses that support a learner in achieving a number of learning goals, such as discovering new content ("discover" in short), training specific competencies and simulating exams. For these learning goals, Paigos generates complete courses which contain all the learning material required by a learner to achieve the goals. Paigos is also used to retrieve single elements that specifically fulfill a given purpose, such as presenting an example or exercise adequate for the current the learner. This functionality is important for remedial, e. g., if a learner fails to solve an exercise, then the presentation of an example might help the learner to overcome the difficulty.

Paigos was developed in a moderate constructivist context. Moderate constructivism emphasizes the active engagement of the learner and stresses that knowledge cannot be transferred from the teacher to the learner but is the result of cognitive processes in the learner's mind. A moderate constructivist learning environment has to stimulate and create opportunities for these processes. Paigos is also based on the concept of "competencies", which advocates that different competencies together build up mathematical literacy [31]. In the following, we describe the Artificial Intelligence framework employed by Paigos and how it is used to describe two scenarios ("discover" and "guided tour"). In total, ActiveMath comprises seven different course generation scenarios.

3.6.1. Hierarchical Task Network Planning

Paigos uses AI planning as a framework for implementing the pedagogical knowledge of how to generate a course. In Hierarchical Task Network planning

(HTN) [29], the goal of the planner is to achieve a list of tasks, where each task is a symbolic representation of an activity to be performed. The planner formulates a plan by using methods to decompose these top tasks into smaller subtasks until primitive tasks are reached that can be carried out directly using operators. The planning operators describe various actions that can be performed directly. Methods describe various possible ways of decomposing non-primitive tasks into subtasks. These are the "standard operating procedures" that one would normally use to perform tasks in the domain. Each method may have constraints that must be satisfied in order to be applicable. Planning is done by applying methods to non-primitive tasks to decompose them into sub-tasks, and applying operators to primitive tasks to produce actions. If this is done in such a way that all of the constraints are satisfied, then the planner has found a solution plan; otherwise the planner will need to backtrack and try other methods and actions.

For course generation, PAIGOS takes a goal task as input and returns a structured sequence of learning object identifiers (basically a table of contents) as a result [37]. The goal task consists of an educational objective that represents the type of learning goal and of learning objects identifiers for which this learning goal should be achieved. As an example, the goal task "discover average slope" represents the goal of a learner who want to reach an in-depth under-standing of the mathematical concept `average slope`. We now take a closer look on the knowledge that formalizes how to achieve such a learning goal.

3.6.2. Course Generation Scenarios

The scenario "discover" generates courses that contain those Learning objects that support the learner in reaching an in-depth understanding of the concepts given in the goal task. The basic structure of the scenario follows a course of action in a classroom that consists of several stages that typically occur when learning a new concept [42]. For each stage, the course contains a corresponding section. The following sections are created:

- Description: describes the aim and structure of a course. Then, for each concept given in the goal task, the following additional sections are created:
- Motivation: motivates the usefulness of the concept using adequate LOs (the precise meaning of an "adequate" LOs is formalized in methods like described below). It also contains the unknown prerequisites.
- Develop: presents the concept and illustrates how it can be applied.
- Train: provides opportunities to train the concept.
- Connections: illustrates the connections between the current and related concepts.
- Reflection: each course closes with a reflection section, which provides the learner with opportunity to reflect on the content presented in the course.

For this scenario, the course generation is started with the single goal task `(discover (`c_1 `...`c_n`))` with $c_1 \ldots c_n$ representing the goal concepts the student wants to learn. A first method, not shown here, inserts a subtask `(learnConceptDiscover (`c_n`))` for each of the concepts. This task is decomposed by the following method:

```
(:method (learnFundamentalDiscover ?c)
         ()
         ((!startSection)
          (introduceWithPrereqSection ?c)
          (developFundamental ?c)
          (proveSection ?c)
          (practiceSection ?c)
          (showConnectionsSection ?c)
          (!endSection)))
```

The keyword `:method` starts the definition of a new method. The subsequent expression `(learnConceptDiscover ?c)` represent the task the method is applied on (expressions starting with "?" stand for variables and are instantiated at run-time). The task is decomposed into 7 subtasks `(!startSection)` ... `(!endSection)`: first, a new section is started, in this case the course itself. Then, several subtasks are inserted that mimic the informal description above. The last subtask closes the course. Similar methods exist for the other tasks.

About 30 methods encode the pedagogical knowledge how to select exercises that are "appropriate" for the learner. The exact meaning of "appropriate" differs depending on the individual learner. The most relevant factors determining exercise selection are the educational level of the learners and their current competency level. In general, the learning context of each learning object should correspond to the educational level of the learner. Otherwise it may be inadequate, that is, either too simple or too difficult (think of a second year university student being presented a definition for elementary school). In addition, in most cases, resources presented to the learner should be of a competency level that corresponds to the learner's since these are the resources he is able to master. In some situations resources with a higher competency level need to be selected, e.g., when aiming at increasing the compe-tency level. The methods also take information about motivation and anxiety into account (assed using the performance in exercises). A similar complex set of methods determines the selection of examples.

The scenario "discover" is based on competencies and competency levels. Since competency-based pedagogy is a relatively novel approach, it is not yet widespread and most of the existing learning environments follow the traditional mastery-based paradigm. With this in mind, we developed the scenario "guided tour" based on Merrill's "First principles of instruction" [26]. In this scenario, for each concept given in the goal task, and for each unknown prerequisite concepts, the following sections are created: An introduction that arises a learner's interest by presenting LOs of the type introduction; a section that presents the concept itself, a section that provides explaining and deepening information about the concept; a section that provides opportunities for the learner to examine demonstrations of applications of the concepts; a section that enables a student to actively apply what he has learned; and finally a section that contains concluding information about the concept.

3.6.3. Course Generation as a Web-Service

The course generator is available as a Web-service (CGWS). The CGWS provides two main kinds of interfaces: the core interface that provides the methods for the course generation, and the repository integration interface that allows a client to register a repository at the CGWS. The core interface consists of the following methods:

- The method `getTaskDefinitions` is used to retrieve the pedagogical tasks which the course generator can process.
- The method `generateCourse` starts the course generation on a given task. The client can make information about the learner available in two ways: if the learner model contains information about the specific learner, then the client passes the respective learner identifier as a parameter. In case no learner model exists, a client gives a list of property-value pairs that is used by the CGWS to construct a temporary "learner model". The course generator performs the planning in the same way as with a real learner model, however its access of learner properties is diverted by the CGWS and answered using the map. Properties not contained in the map are answered with a default value.

The result of the course generation is a structured sequence of educational resources represented in an IMS Manifest. Since the returned result does not contain the resources but only references, the return result is not an IMS CP.

The interface for repository registration consists of the following methods:

- The method `getMetadataOntology` informs the client about the metadata structure used in CGWS. It returns the ontology of instructional objects described in Section 3.4.
- The method `registerRepository` registers the repository that the client wants the course generator to use. The client has to provide the name and the location (URL) of the repository. Additional parameters include the ontology that describes the metadata structure used in the repository and the mapping of the OIO onto the repository ontology.
- The method `unregisterRepository` unregisters the given repository.

3.6.4. Client Interfaces

A client that wants to use the CGWS needs to provide information about the educational resources as well as about the learner (if available).

The interface `ResourceQuery` is used by the mediator to query the repository about properties of educational resources. The interface consists of the following methods (the same as described in Section 3.4.3):

- `queryClass` returns the classes a given resource belongs to.
- `queryRelation` returns the set of identifiers of those educational resources the given resource is related to via the given relation.
- `queryProperty` returns the set of property-value pairs the given resource has.

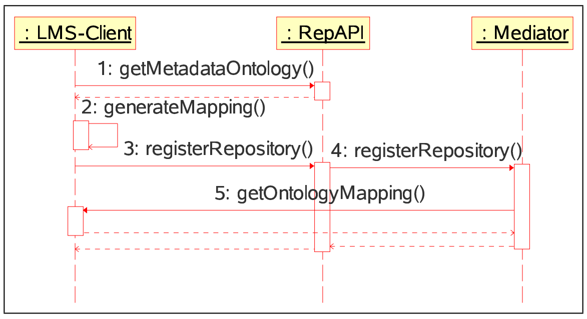


Figure 4. A sequence diagram illustrating the repository registration

The *LearnerPropertyAPI* makes the learners' properties accessible to the CGWS in case the client contains a learner model and wants the course generator to use it. In the current version of the CGWS, this interface is not yet implemented. It would require a mediator architecture similar to the one used for repository integration.

3.6.5. Interaction between Client and Server

In this section we describe the communication between client and server performed when registering a repository and for course generation.

A repository is registered in the following way (for a sequence diagram illustrating the registration, see Figure 4): in a first step, the client (LMS-Client in the figure) retrieves the metadata ontology used in the CGWS (i. e., the OIO). The ontology is then used to generate a mapping between the OIO and the ontology representing the client metadata (Step 2) (the currently existing mappings were manually authored). Then, the repository is registered using the method `registerRepository` (Step 3). The repository is added to the list of available repositories and made known to the mediator (Step 4). Subsequently, the mediator fetches the ontology mapping from the client and automatically generates a wrapper for querying the `contentAPI` of the client.

A client starts the course generation using the service method `generateCourse`. In a first step, the CGWS checks whether the task is valid. If so, the course is generated by the course generator. During the generation process, PAIGOS sends queries to the mediator, which passes the queries to the repository. Like in ACTIVEMATH, the results are cached. After the course is generated, the `omgroup` generated by PAIGOS is transformed into an IMS manifest and sent to the client.

The CGWS is still in an early stage of development and further work is necessary to realize a mediator-like architecture for the generic integration of learner models. Yet, despite being a prototype, the CGWS was successfully used by several third-party systems, e. g., MATHCOACH (a learning tool for statistics [10]) and TEAL (workflow embedded e-learning at the workplace [33]).

3.7. RICH CLIENT ACCESS TO ACTIVEMATH

In this section, we will give an example of rich client applications in ACTIVEMATH that uses several of the service described above (the knowledge base and the course generator).

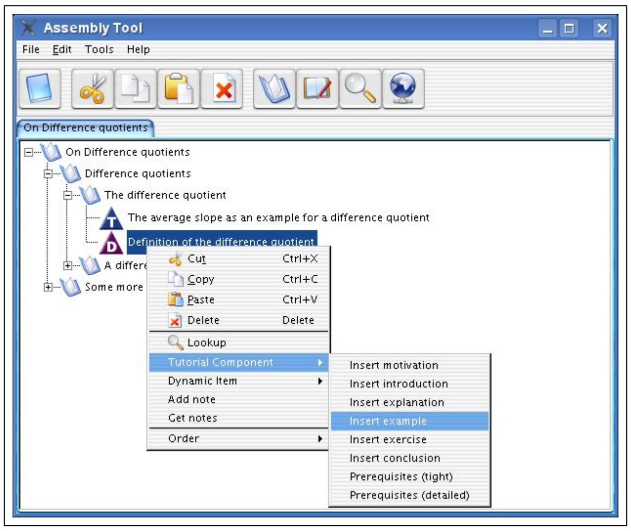


Figure 5. Screenshot of the assembly tool

3.7.1. ActiveMath's Assembly Tool

ActiveMath's *assembly tool* allows a learner to create a book on her own by dragging educational resources from ActiveMath (but also any other resource addressable by an URI). The tool was designed to support the learner's meta-cognitive reasoning, self-regulated learning and active engagement with the content.

The principal actions supported by the assembly tool are the creation of structured courses by adding chapters and drag-and-drop of resources into a table of contents. In addition, a learner has access to Paigos's functionality using a context menu. She can select the direct insertion of resources that fulfill a pedagogical task specified by the learner or insert a dynamic task that is achieved at a later time. Figure 5 contains a screenshot that illustrates the integration. In the example, the learner uses the course generator to select an example for "the definition of the difference quotient".

The Assembly Tool is started via Java WebStart. Upon clicking on a link to the Assembly Tool ActiveMath generates dynamically a JNLP [12] file and sends it back to the browser. The browser invokes Java WebStart and passes it the file. It retrieves them and starts a new Assembly Tool process. The file contains information where to locate the assembly tool's binaries, general information about the application as well as context-specific information supplied by the ActiveMath server.

Since ActiveMath is a user-adaptive system, the Assembly Tool receives context-dependant information in the JNLP file properties. The information is provided at launch time by instantiating service wrappers and providing their URLs as properties at launch time.

Both the URL to the content base and to the user manager are secured by a random number. This way, malicious alterations, in particular of the user model, are prevented. In addition, the URL to the ActiveMath server is provided. For

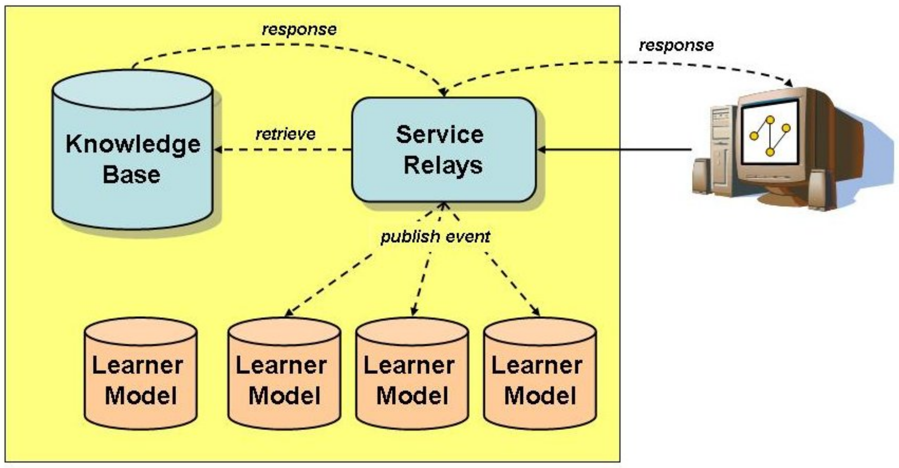


Figure 6. Assembly Tool communication with ActiveMath

the Assembly Tool to address the current user, the ActiveMath server includes the current user's identifier, name, and default language. If the user intends to edit an own book a book identifier is added.

The Assembly Tool is able to interoperate with other ActiveMath components. Messages are transported via xml-rpc.

Communication between Assembly Tool and LeActiveMath is twofold (see Figure 6). Requests to the content knowledge base are synchronous while client-events are issued asynchronously and they are distributed via ActiveMath's event framework.

Any action in the Assembly Tool that alters a book the student is currently working with triggers an event that is sent to the ActiveMath server. Also, actions that call service methods as well as upload/save/open/new actions cause an event.

Finally, the claim for consistency of appearance and language sketched above is achieved by the following interoperability result. Each item presented in the web-browser (e.g. within a book) can have its title dragged and dropped into the assembly tool so that the corresponding item is included in the designated page. This usage relies on the semantics of web-links which uniquely identified the rendered content items. Dropping such an item in the assembly tool triggers the insertion of the item in the table-of-contents but also a call to the knowledge base to determine the type and title of the item thus yielding almost the same rendering of the item's title in a completely different setting.

3.7.2. Integrated Semantic Mathematical Input and Output

A similar principle has been approached for the mathematical output (that is, the rendering of formulæ) and input (that is, where the user inputs formulæ): the OpenMath representation is used as lingua franca of any exchange between the components, clients and servers. Its rendering produces consistent notations which adapt to the local context (e.g. language, domain-tradition, for example the presentation of the binomial coefficient being C_a^b, C_b^a or $\binom{a}{b}$ in France, Russia, or Great Britain).

This rendering is specified by authorable notations which are the main source of knowledge for the conversion of OpenMath expressions to presentation languages. The presentation of mathematics is consistent throughout ActiveMath, regardless of whether a formula is rendered in the ActiveMath itself or other tools such as the input editor. This editor [20] is run as a Java applet and allows writing of formulæ within the interactive exercises, the free computer-algebra-system, and the mathematical search tool. It also enables drag-and-drop of formulæ into a graph plotter.

The input-editor, at time of launch, is allocated a *server-side-storage*, i. e., a URL where it can read from and post to. This allows its content to be drag-and-droppable to other input-editor-instances, prevents the usage of the fragile JavaScript↔Java communication, and avoids the OpenMath expression to clutter the URL.

Copy&paste is an important help for learners to input mathematical formulæ. Thus, in ActiveMath every term of a mathematical expression can be dragged-and-dropped into an input-editor applet. Behind the scenes, a URL is drag-and-dropped. This URL is a *clip-URL* which provides the source OpenMath representation, or reformulations of them. The input editor then requests such a URL and inserts the OpenMath representation thus obtained in the formula where the drop occurred. See [17] for more details about the transfer facility of ActiveMath.

The integrated nature of the mathematical input-and-output was evaluated positively by students. Due to technical constraints, initiating the drag-and-drop required a slightly more complicated interaction than traditional drag-and-drop. This was criticized as being unconventional and unexpected. The web-nature of our ActiveMath system could, however, not do more than a browser showing untrusted pages could. A standardization at the level of browser is thus needed, the W3C WebAPI working group may address it.[3]

3.7.3. Other Client Tools

Several other client tools are running using a similar architecture as the assembly tool. They all are contextualized at launch time with service wrappers either using JNLP or applets. One of them is the Interactive Concept Mapping Tool iCMaP [24]. This tool allows learners to drop item references and organize them in concept-maps which can be validated against the knowledge base.

Other tools are used in exercise situations and send exercise events to the event dispatcher which allows the learner model to be kept up-to-date. Finally, a desktop based exercise system for combinatorics is currently being investigated for integration. Such a system would not be launched by an applet or JNLP file but by the download of an exercise file whose type is associated to the application.

[3] `http://www.w3.org/2006/webapi/`.

4. RELATED WORK

A large number of research groups investigate educational services based on semantics in an educational context. However, to our knowledge, no other system covers such a broad and complete set of functionalities as ActiveMath.

For instance, only few systems (e.g., [7]) offer a semantic representation of the content, hence loosing the possibility of rendering different output format or using the content as input for tools such as Computer Algebra Systems. To our knowledge, semantic-based copy-and-paste of formulæ is still an exclusivity of ActiveMath.

However, most systems use XML-based metadata, based on and extending IEEE LOM. Several groups, unsatisfied by the limited pedagogical coverage of LOM worked on metadata expressive enough for pedagogically mature systems (e. g., [18]). The ontology of instructional objects described in Section 3.3 draws on these approaches and represents the minimal pedagogical information necessary for pedagogically founded learner support.

Adaptive Hypermedia (AH, see [4]) covers techniques of adapting hypermedia objects with respect to the user. Well known systems are ELM-ART [40] and AHA [1]. ActiveMath offers several AH techniques but is more flexible as complete courses can be generated based on pedagogical knowledge. Course Generation (or Adaptive Trail Generation) has been investigated since long [39,38,35,34] and is still an active area of research. For instance, [25,15,13,14] focus on the generation of learning trails and courses. However, these approaches do not allow for such detailed representation of pedagogical knowledge as done in ActiveMath. Most rely on the sequencing of full pages which is a crucial loss of re-usability: typically, a definition can be shared among several learning-contexts but it is important for learning purposes to differentiate the introductory, example, and exercise material (for example relying on real world examples for the given audience).

In the last years, semantics and services their relevance for education was investigated by several groups. [30] describe an RDF-based peer to peer networking infrastructure. [27] describe an approach to query Semantic Web resources by querying and transforming RDF models using a rule language based on Horn logic. Their focus lies on the general approach, thus the e-learning scenario provided as an example in their paper is limited to querying exercises and it appears that such a performance as we expect is not to be expected in a near future.

LESSONS LEARNED AND CONCLUSION

In this chapter, we described educational services in ActiveMath and the underlying service-oriented technologies we are using. During the development of ActiveMath, we experimented with different technologies. The lesson we have learned is that being generic and abstract is nice, but speed matters more. More specifically this means that we used the most primitive but efficient technology that was available and appropriate to the specific service (e. g., XML-RPC or Java for internal communication and Web services for interaction with external components). In early versions of ActiveMath, we made extensive use of its distribution features: the learner model ran on server a, the knowledge base on server b, etc. While this lowered the load on each single machine, the overall overhead increased dramatically. In real usage, that is, in a classroom setting involving several dozens of students (as done for the evaluation performed in the LeActiveMath project), the distribution was simply too inefficient and eventually we moved all internal ActiveMath components back into a single server.

One example that illustrates the load during real application is course generation. Generating a expanded course for a single concept (the course consists of a total of 40 resources) results in about 1 500 mediator queries that are expanded to more than 11 000 queries to the repository. On the other end of the spectrum, generating an expanded course for 20 fundamentals (the course consists of a total of 370 resources) results in 21 500 mediator queries and more than 100 000 expanded queries. While the amount of queries can be reduced to 150/1 200 and 2 700/14 100 by expanding some goals on-demand-only [37], the numbers remain significant. As a consequence, the course generator's performance considerably depends on the repositories and the learner model. In case the components reside on different servers, the very network latency alone reduces the overall performance: the LeActiveMath exercise repository is located in Eindhoven, the Netherlands. When accessed from Saarbrücken, Germany, it answers a single query in about 80 milliseconds. As a consequence, the generation of a 4 concepts course that requires 3 300 queries requires 4:30 minutes instead of 290 milliseconds. On the Web, four minutes are an eternity. Few learners will wait patiently for the course to appear

We think that while service- and grid-oriented educational computing has its benefits, we need to be aware of the short-comings that it still exhibits today hence should only use it where the benefits pay off.

References

[1] Ad Aerts, David Smits, Natalia Stash, and Paul De Bra. AHA! Version 2.0, more adaptation flexibility for authors. In Griff Richards, editor, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2002*, pages 240–246, Montreal, Canada, 2002. AACE.

[2] L. Aroyo and R. Mizoguchi. Authoring support framework for intelligent educational systems. In U. Hoppe, F. Verdejo, and J. Kay, editors, *Proccedings of AI in Education, AIED-2003*, pages 362–364. IOS Press, 2003.

[3] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. Web Services Architecture Requirements. Online: `http://www.w3.org/TR/wsa-reqs/`, 2004.

[4] Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User Adapted Interaction*, 11(1/2):87–110, 2001.

[5] Peter Brusilovsky and Julita Vassileva. Course sequencing techniques for large-scale web-based education. *International Journal of Continuing Engineering Education and Life-long Learning*, 13(1/2):75–94, 2003.

[6] Stephen Buswell, Olga Caprotti, David Carlisle, Mike Dewar, Marc Gaëtano, and Michael Kohlhase. The openmath standard, version 2.0. Technical report, The OpenMath Society, June 2004. Available at `http://www.openmath.org/`.

[7] Hans Cuypers and Hans Sterk. Mathbook, web-technology for mathematical documents. In *Proceedings of the BITE 2001 conference, Eindhoven, The Netherlands, 22-24 November*, 2001. See also from `http://www.riaca.win.tue.nl/`.

[8] J. de Bruijn, D. Foxvog, and K. Zimmermann. Ontology Mediation Patterns Library V1. D4.3.1, SEKT-project, February 2005.

[9] G. Goguadze, A. González Palomo, and E. Melis. Interactivity of Exercises in ActiveMath. In *In Proceedings of the 13th International Conference on Computers in Education (ICCE 2005)*, pages 107–113, Singapore, 2005.

[10] Barbara L. Grabowski, Susanne Gäng, Jörg Herter, and Thomas Köppen. MathCoach und LaplaceSkript: Ein programmierbarer interaktiver Mathematiktutor mit XML-basierter Skriptsprache. In Klaus P. Jantke, Klaus-Peter Fähnrich, and Wolfgang S. Wittig, editors, *Leipziger Informatik-Tage*, volume 72 of *LNI*, pages 211–218. GI, 2005.

[11] Yusuke Hayashi, Jacqueline Bourdeau, and Riichiro Mizoguchi. Ontological support for a theory-eclectic approach to instructional and learning design. In Wolfgang Nejdl and Klaus Tochtermann, editors, *Innovative Approaches for Learning and Knowledge Sharing, First European Conference on Technology Enhanced Learning, EC-TEL 2006, Crete, Greece, October 1-4, 2006, Proceedings*, volume 4227 of *LNCS*, pages 155–169. Springer, 2006.

[12] Java Software (Sun MicroSystems, Inc.). Java (tm) network launching protocol & api specification (jsr-56), 2004. Available at `http://java.sun.com/products/javawebstart/download-spec.html`.

[13] Pythagoras Karampiperis and Demetrios Sampson. Automatic learning object selection and sequencing in web-based intelligent learning systems. In Zongmin Ma, editor, *Web-Based Intelligent e-Learning Systems: Technologies and Applications*, chapter III, pages 56–71. Information Science Publishing, 2005.

[14] Ian O. Keeffe, Aoife Brady, Owen Conlan, and Vincent Wade. Just-in-time generation of pedagogically sound, context sensitive personalized learning experiences. *International Journal on E-Learning*, 5(1):113–127, 2006.

[15] K. Keenoy, A. Poulovassilis, G. Papamarkos, P.T. Wood, Vassilis Christophides, Aimilia Maganaraki, Miltiadis Stratakis, P. Rigaux, and N. Spyratos. Adaptive personalisation in self e-learning networks. In *Proceedings of First International Kaleidoscope Learning Grid SIG Workshop on Distributed e-Learning Environments*, Napoly, Italy, 2005.

[16] Michael Kohlhase. *OMDoc – An Open Markup Format for Mathematical Documents.* Springer Verlag, 2006.

[17] Paul Libbrecht and Dominik Jednoralski. Drag and Drop of Formulae from a Browser. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[18] Ulrike Lucke, Djamshid Tavangarian, and Denny Voigt. Multidimensional educational multimedia with ml3. In Griff Richards, editor, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2003*, pages 101–104, Phoenix, Arizona, USA, 2003. AACE.

[19] Tiziana Margaria. Service is in the eyes of the beholder. *Computer*, 40(11):33–37, 2007.

[20] Daniel Marquès, Ramon Eixarch, Glòria Casanellas, Bruno Martínez, and Tim Smith. WIRIS OM Tools a Semantic Formula Editor. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[21] E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A Generic and Adaptive Web-Based Learning Environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

[22] E. Melis, J. Büdenbender, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Knowledge representation and management in activemath. *Annals of Mathematics and Artificial Intelligence, Special Issue on Management of Mathematical Knowledge*, 38(1-3):47–64, 2003. Volume is accessible from `http://monet.nag.co.uk/mkm/amai/index.html`.

[23] E. Melis, G. Goguadze, M. Homik, P. Libbrecht, C. Ullrich, and S. Winterstein. Semantic-Aware Components and Services of ActiveMath. *British Journal of Educational Technology*, 37(3):405–423, may 2006.

[24] E. Melis, P. Kärger, and M. Homik. Interactive Concept Mapping in ActiveMath (iCMap). In Jörg M. Haake, Ulrich Lucke, and Djamshid Tavangarian, editors, *Delfi 2005: 3. Deutsche eLearning Fachtagung Informatik*, volume 66 of *LNI*, pages 247–258, Rostock, Germany, sep 2005. Gesellschaft für Informatik e.V. (GI).

[25] Néstor Darío Duque Méndez, Claudia Jiménez Ramírez, and Jaime Alberto Guzmán Luna.

IA planning for automatic generation of customized virtual courses. In *Frontiers In Artificial Intelligence And Applications, Proceedings of ECAI 2004*, volume 117, pages 138–147, Valencia (Spain), 2004. IOS Press.

[26] M. D. Merrill. First principles of instruction. *Educational Technology Research & Development*, 50(3):43–59, 2002.

[27] Z. Miklos, G. Neumann, U. Zdun, and M. Sintek. Querying semantic web resources using TRIPLE views. In *The SemanticWeb - ISWC 2003, Sanibel Island, USA*, volume 2870 of *Lecture Notes in Computer Science*, pages 517–532, Heidelberg, Germany, 2003. Springer.

[28] R. Mizoguchi and J. Bourdeau. Using ontological engineering to overcome AI-ED problems. *International Journal of Artificial Intelligence in Education*, 11(2):107–121, 2000.

[29] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[30] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. Edutella: a P2P networking infrastructure based on RDF. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 604–615, New York, NY, USA, 2002. ACM Press.

[31] OECD, editor. *Learning for Tomorrows World — First Results from PISA 2003.* Organisation for Economic Co-operation and Development (OECD) Publishing, 2004.

[32] Darwyn R. Peachy and Gordon I. McCalla. Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies*, 24(1):77–98, 1986.

[33] Oleg Rostanin, Carsten Ullrich, Harald Holz, and Shenwei Song. Project teal: Add adaptive e-learning to your workflows. In Klaus Tochtermann and Hermann Maurer, editors, *Proceedings: I-KNOW'06, 6th International Conference on Knowledge Management*, pages 395–402, Graz, Austria, Sep 2006.

[34] Cornelia Seeberg, Klaus Reichenberger, Stephan Fischer, Ralf Steinmetz, and Achim Steinacker. Dynamically generated tables of contents as guided tours in adaptive hypermedia systems. In Piet Kommers and Griff Richards, editors, *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 1999*, pages 640–645, Chesapeake, VA, 1999. AACE.

[35] M. Specht and R. Oppermann. ACE - adaptive courseware environment. *The New Review of Hypermedia and Multimedia*, 4:141–162, 1998.

[36] IEEE Learning Technology Standards Committee. 1484.12.1-2002 IEEE standard for Learning Object Metadata, 2002.

[37] Carsten Ullrich. *Course Generation as a Hierarchical Task Network Planning Problem.* PhD thesis, Computer Science Department, Saarland University, Saarbrücken, 2007.

[38] Kris Van Marcke. GTE: An epistemological approach to instructional modelling. *Instructional Science*, 26:147–191, 1998.

[39] Julita Vassileva. Dynamic Courseware Generation: at the Cross Point of CAL, ITS and Authoring. In *Proceedings International Conference on Computers in Education, ICCE'95*, pages 290–297, Singapore, 1995.

[40] G. Weber and Peter Brusilovsky. ELM-ART: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12(4):351–384, 2001.

[41] G. Wiederhold. Mediators in the architeture of future information systems. *The IEEE Computer Magazine*, March 1992.

[42] Friedrich Zech. *Grundkurs Mathematikdidaktik.* Beltz Verlag, Weinheim, 2002.