MultiMedia Instruction in Safe and Secure Systems*

Bernd Krieg-Brückner[1], Dieter Hutter[2], Arne Lindow[1], Christoph Lüth[1], Achim Mahnke[1], Erica Melis[3], Philipp Meier[6], Arnd Poetzsch-Heffter[4], Markus Roggenbach[1], George Russell[1], Jan-Georg Smaus[5], Martin Wirsing[6]

[1] Bremen Institute for Safe and Secure Systems, Universität Bremen
[2] DFKI Saarbrücken
[3] Universität des Saarlandes
[4] Universität Kaiserslautern
[5] Universität Freiburg
[6] Ludwig-Maximilians-Universität München

Abstract. The aim of the MMiSS project is the construction of a multimedia Internet-based adaptive educational system. Its content will initially cover a curriculum in the area of Safe and Secure Systems. Traditional teaching materials (slides, handouts, annotated course material, assignments, and so on) are to be converted into a new hypermedia format, integrated with tool interactions for formally developing correct software; they will be suitable for learning on campus and distance learning, as well as interactive, supervised, or co-operative self-study. To ensure "sustainable development", i.e. continuous long-term usability of the contents, coherence and consistency are especially emphasised, through extensive semantic linking of teaching elements and a particular version and configuration management, based on experience in formal software development and associated support tools.

1 Introduction and Overview

In the last few years the area of *safe and secure systems* has become more and more important. Software is increasingly used to control safety-critical embedded systems, in aeroplanes, spaceships, trains and cars. Albeit its associated security risks, electronic commerce over the internet is rapidly expanding. This requires a better training of computer scientists in the foundations and practical application of formal methods used to develop these systems. The aim of the MMiSS-project (*MultiMedia instruction in Safe and Secure Systems*) is to set up a multimedia internet-based adaptive educational system, covering the area of Safe and Secure Systems. With a consistent integration of hypermedia course materials and formal programming tools, teaching in this area will attain a level hitherto impossible in this form. The system will be as suitable for

* The MMiSS project has been supported by the German Ministry for Research and Education, bmb+f, in its programme "New Media in Education".

learning on campus and for distance-learning with its associated management of assignments, as it is for interactive, supervised, or co-operative self-study.

At the core of the system is the hypermedial adaptation of a series of courses or lectures on the development of reliable systems. The teachers should be able to store various sorts of course material, such as overhead slides, annotations, lecture notes, exercises, animations, bibliographies, and so on, and retrieve them again for use in teaching, notably also re-using material of other authors. The system provides a formal framework for the integration of teaching materials based on a *semantic structure (ontology)* and enables fast directed access to individual teaching elements.

An initial collection of courses is already available and should be further hypermedially developed as part of the project in an Open Source Forum (cf. Sect. 8). It covers the use of *formal* methods in the development of (provably) *correct* software. Highlights include data modelling using algebraic specifications; modelling of distributed reactive systems; handling of real-time with discrete events; and the development of hybrid systems with continuous technical processes, so-called *safety-critical systems*. The curriculum also covers informal aspects of modelling, and introduces into the management of complex developments and into the basics of *security*.

The teaching material should, where possible, be available in several different *variants*. It should be left to the teachers, or the students, to choose between variants, according to the educational or application context. For example reactive systems could be modelled with either process algebras or Petri-nets; the material could be avaliable in English or other natural languages. The system also contains meta-data, representing ontological, methodological and pedagogical knowledge about the contents.

An important educational aspect is to teach about the possibilities and limits of formal tools. *Tools for formal software development* should be integrated in the system, to illustrate and intensify the contents to be taught. Thus students doing assignments can use the system to test their own solutions, while gathering experience with non-trivial formal tools. The integration of didactic aspects with formal methods constitutes a new quality of teaching. It will become possible both to present a variety of formal tools as a subject for teaching, and to use them as a new medium. Thus an algorithm can for instance be simultaneously developed, presented, and verified.

The goal of applying the MMiSS-system in as many universities and companies as possible, and the fact that the area of Safe and Secure Systems will further evolve in the future, requires the highest level of *flexibility, extensibility and reusability* of the content. It should be possible to incrementally extend or adapt content and meta-data, to suit the teacher's individual requirements, and to keep them up-to-date. We expect the system to be easily adapted and well usable in other subject domains.

As the individual parts of a curriculum rely on each other, there is a network of *semantic dependencies* that the system should be able to administer; at the least it has to offer a version- and configuration management. Additionally, an

ontology allows a better support for orientation and navigation within the content. It forms the basis for adaptation to the user, for example by learning from exercises which concepts the students have understood, and by adapting future assignments accordingly.

The formalisation of semantic dependencies means that the system can help to *maintain the consistency (and completeness)* of the content. Definitions must be coordinated to suit each other; the removal or adaptation of some material may force the removal or adaptation of all dependent concepts. In formal software development, a similar problem has to be solved: there are also semantic dependencies between different parts of a development, for example between specification and implementation. Some of the project partners have already developed techniques for the administration of such dependencies as things change, and implemented them in development tools. Here we perceive an important *synergy* between expertise in formal software development – and support tools – and the demands of long-term *sustainable development for re-use* of consistent multimedia materials in an efficient and productive educational system.

Outline. Although the MMISS project is concerned with the development of a multi-media based eduction system for safe and secure systems, the techniques and tools developed in this project are not restricted to this particular area. In this sense we start with a description of general concepts to structure documents according to their semantics in Sect. 2 and briefly sketch an extension of LaTeX, called MMiSSLaTeX, that allows the specification of these additional structuring concepts. Sect. 3 describes the particular ontology used to structure the contents in this particular problem domain. Sect. 4 illustrates the tool support of MMISS to create or maintain such course material. MMISS provides various authoring tools to transfer LaTeX or Powerpoint based course material, enriched with additional semantic information, into the MMISS-Repository which maintains versions, configurations and the (user-defined) consistency of the material. Sect. 7 focusses on the presentation of the teaching material stored in the Repository. MMiSS supports two presentation mechanisms, one simply using the layout information as it is encoded in input documents written in MMiSSLaTeX, and the ActiveMath environment that dynamically generates the presentations according to the skills and needs of the user. We conclude this presentation in Sect. 8 with a prelimary evaluation of teaching experiences using the MMiSS tools and with a view towards future developments.

2 Structuring Mechanisms for Documents

MMiSS aims at the support of the creation, the maintenance and the presentation of education material dedicated to various courses or lectures in a domain. As an author, one has to be aware of the various (mainly semantically oriented) structuring mechanisms hidden in these documents. Writing, for instance, a mathematical document in LaTeX, there is an explicit syntactical structure of the document triggered by LaTeX commands such as `section`, `paragraph`, etc.

Additionally, there are other more semantically oriented structuring mechanisms. Defining mathematical entities, we are likely to build up a hierarchy of definitions. In a conventional LaTeX document, we do not represent these relations explicitly. However, we have to keep them in mind once we want to change documents in a consistent way. The overall design of MMiSS aims at an explicit representation of such relations, e.g.

- to navigate in the material along the semantical relations: during class, as a teacher; after class or during self-study, as a student;
- to support maintenance and update of course material.

2.1 An Ontology of Users

Ontologies provide the means for establishing a semantic structure. An ontology is a formal explicit description of concepts in a domain of discourse. Ontologies are becoming increasingly important because they provide also the critical semantic foundations for many rapidly expanding technologies such as software agents, e-commerce and knowledge management. Ontologies consist of concepts and relations between these concepts. Properties of a concept are specified by describing its various features and attributes. As instantiation we use a subset of the modeling language UML which is an actual de facto standard language for software development. As an ontology describes domain concepts abstractly by means of classes, subclasses and slots, UML seems to be particularly well-suited for the diagrammatic representation of the ontology [5].

Before we explore the variety of MMiSS Document Constructs for structuring in the sequel, let us consider a little example of an ontology in Fig. 1; for a more extensive treatment of the ontology subject see Sect. 2.6 and Sect. 3 below.

The example shows an *ontology* of potential Users of MMiSS and their Roles, resp. A *Professor*, as an *Academic User*, may assume the Role of a *Teacher*, thereby only having reading access to the material, or of an *Author* with a particular kind of writing access. This ontology is of course much simplified (there are also Developer and Administrator Roles, etc.). As notation we use a subset of UML class and object diagrams. It shows, however, the general principles:

- a taxonomic hierarchy of *classes* (the fat arrow with a triangular head denotes the "subclassOf" relation), e.g. a Professor is an Academic User, inheriting all its properties,
- individuals ("*objects*") of these classes (not shown here, but cf. _deAttribute_ in Fig. 3 as an object of class LanguageAttribute, distinguished in the notation by underlining or leading and trailing underscores), and
- a (hierarchy of) *relations* (called associations in UML), declared between the classes and applied to objects, e.g. User assumesRole Role. Note that relations are inherited by the classes involved, e.g. Professor assumesRole Teacher.

We will use this notation to define and illustrate (parts of) the MMiSS ontology in the sequel. As an experiment in formatting, classes, objects and relations relating to the ontology used in this paper are highlighted by special fonts when

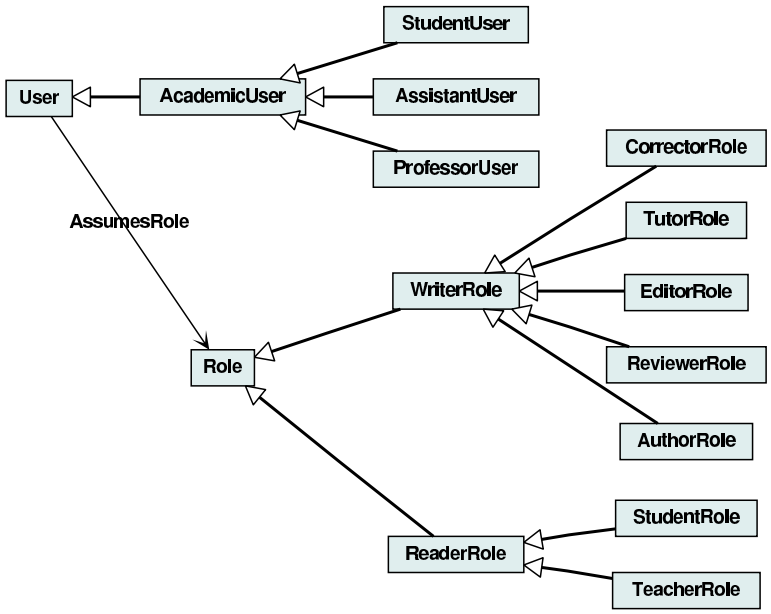

Fig. 1. Ontology of Users and Roles.

referred to in the text (as in ontology), and classes and objects pertaining to the MMiSS ontology are capitalised (as in LanguageAttribute).

2.2 Document Structure

Structural Entities. The primary purpose of structuring documents is conceptual. We are used to textually nesting paragraphs in sections, and sections in other sections, possibly classified into (sub)subsections, chapters, parts of documents or the like. The same is true here, cf. Fig. 2 that shows part of the ontology of MMiSS *Structural Entities*: *Sections* may be nested; they are not classified as chapters or the like to ease re-structuring without the need for renaming (section numbering etc. is, if desired, done automatically anyway during layout; the title of a Section will appear in a table of contents). A Section may contain smaller nested entities such as Units or Atoms, see below.

The largest Structural Entity is a *Package*. A Package is a document that corresponds to a whole course or book and contains all Structural Entities pertaining to it. A Package contains a Prelude that contains a kind of "global declarations" for it, e.g. a BibliographyPrelude, or an ImportPrelude for other Packages ("structuring in-the-large"), see Sect. 2.3.

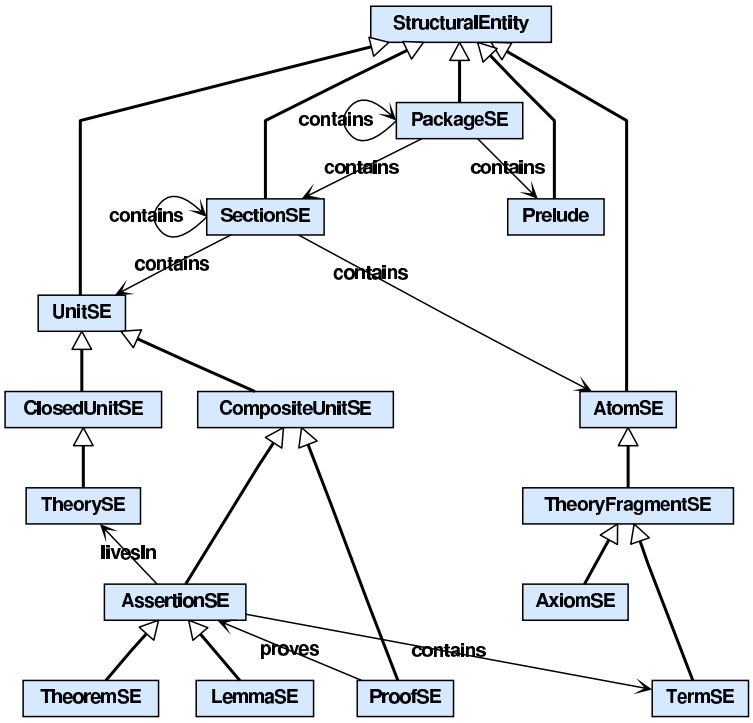

Fig. 2. (Partial) Ontology of Structural Entities.

Ontology Prelude. In particular, it may contain an OntologyPrelude, where the elements of an ontology may be declared (cf. Sect. 2.6) — it acts like a signature of the Package for semantic interrelation, promising these elements to be defined in this Package, such that they become available when imported by another.

Sections, Units and Atoms. Each Section should contain three special sub-Sections or Units: The *Abstract* contains an overview of the Section; the *Introduction* gives a motivation for the content to come and sets a didactic goal ("what we are about to learn"); the *Summary* at the end recalls the highlights of the content ("what we have learned"). Note that there are no explicit "transition" Paragraphs between Sections since they would assume a given order; instead, the Introduction should refer to the upper context ("what we already know"), if necessary, and the Summary should provide forward references to the lower context ("what we will learn more about") in subsequent Sections.

A Section may contain *Unit*s, and Units may contain other Units or Atoms. A Unit is an entity one would like to be able to keep together and eventually present as a whole as far as possible, i.e. on a single slide in a lecture (or a single page in a book), possibly with a continuation slide with the same title. The Unit is the primary structuring facility; it is the minimal context for editing and the primary unit of change (corresponding to a node in the structure graph, see change management in Sect. 6.3).

As we see in Fig. 2, a *ClosedUnit* may be used to classify the enclosed content to be of a particular kind, e.g. a whole Theory in a particular Formalism such as CASL. A *CompositeUnit* may have further internal structure, for example a *List*, a *Table*, a structured *Proof*, or an *Example*.

An *Atom*, such as a *TextFragment* or a *TheoryFragment* (e.g. an *Axiom*), is an indivisible leaf of structuring and the smallest Structural Entity that can be shared (see Sec. 2.2 on structural sharing); it is usually not shown in the structure graph unless a visualisation of the micro-structure of a Unit is explicitly requested.

Conceptual and Formal Structure. The (partial) ontology in Fig. 2 is tailored to the particular application domain of the MMiSS project: safe and secure systems with formal methods. While it is meant to be generally applicable and extensible, it also specially caters for formal, e.g. mathematical, documents. Therefore some of the Units or Atoms may be classified as *formal*; these are associated with a particular Formalism. A *Formalism* comprises, in general, a formal Syntax and (hopefully more often than not) a formal Semantics, cf. also Sect. 3. Examples are a *Program* in a programming language or a *Theory* in a specification language. Thus a formal Atom such as an Axiom, while being atomic from a document structuring point of view, may indeed have further substructure when analysed by a specialised tool.

This way the structure graph contains the formal structure as a subgraph (cf. Fig. 14). A particular document may contain consistent formal sub-documents, e.g. a complete executable Program or a complete Theory, to be analysed together.

Sharing. Formal entities may be embedded piecewise (e.g. just an Axiom of a Theory), as they are being introduced and explained from a conceptual or pedagogical point of view. However, it is also a good idea to present them together, possibly in a separate part of the same document. Thus they are exhibited as a consistent whole, both from a conceptual point of view (e.g. a complete Theory with all TheoryFragments put together) and the technical consideration of having a complete formal document that can be treated by a tool (e.g. analysis of a complete Theory or compilation and execution of a Program with input data). Note also that it is often necessary for pedagogical purposes to be able to present alternatives and variations in a document, even incomplete or intentionally wrong ones that should not be subjected to formal analysis.

Units or Atoms of such a whole formal (sub)document may then be referred to repeatedly in other parts of the document; one would often wish them to be

included as such instead of a mere Reference. Indeed, an entity will often appear in more than one place, e.g. as an Axiom in an explanatory Paragraph and as part of a consistent and complete Theory in an appendix. A copy will not do; common experience dictates that two copies of the "same" entity have a tendency to differ eventually. Thus *structural sharing* is needed, avoiding the danger of unintentional difference: an Axiom named by a Label in one part of a document (or a different document) may be included by an IncludeAtom operation, with a link to this Label, in another.[1] This operation will trigger a textual expansion in the presentation of the document such that both occurrences are indistinguishable in the presentation. In the source, the Axiom has a "home" where it can be edited, whereas it cannot be edited at the positions of the IncludeAtom operation. From a methodological point of view, it is preferable to maintain a complete Theory, which is, however, structured in such a way that links to a particular Axiom are possible from other places.

Sharing is not restricted to formal entities. Indeed, whole sub-documents can be shared when composing a new document from bits and pieces of existing ones.

Comprises and ReliesOn Relations. The textual nesting gives rise to *contains* relations and the include operations to *includes* relations corresponding to arrows in a directed acyclic graph, the structure graph, see Sect. 5.4 (cf. also Fig. 2 and Fig. 14). An Axiom, for instance, is contained in a Section, while Sections are themselves contained in Packages. MMiSS defines a hierarchy of Structural Entities to define the contains relation, e.g. Packages, Sections, Units, or Atoms. The contains and includes relations are special cases of the *comprises* relation; in the sequel we will make use of this comprises relation to define an appropriate change management for MMiSS-documents.

Besides the comprises relations, there is a family of *reliesOn* relations, reflecting various semantic dependencies between different parts of a document (cf. Fig. 2). For example, an Assertion (such as a Theorem) *livesIn* a Theory, a Proof *proves* an Assertion, an Example *illustrates* a Definition, and so on. In this case, we would usually like to insist on a linear order of appearance, i.e. the right-hand-side (target) of the relation should (textually) be presented before the left-hand-side.

2.3 Packages

Packages provide a means for modular document development by introducing *name spaces*. When writing a document, authors introduce identifiers as Labels for Structural Entities or as technical terms in an ontology. If these identifiers, subsumed as *names* in the sequel, are defined more than once, we say there is a *name clash*.

A Package encapsulates the name space of a document, such that names defined in a Package do not clash with names from other Packages. In order to use

[1] It is technically immaterial, whether the Axiom appears in the Paragraph and the IncludeAtom operation in the Theory, or vice versa.

names from other Packages, these have to be imported explicitly (see below). In other words, Packages are very much like modules in programming languages such as Modula-2, Haskell, or Java.

Package Hierarchy. Packages are organised in a folder hierarchy, with the names given by *paths*. Because path names can get very long, paths can be renamed by *Path* declarations (*aliases*) of the form

$$\mathsf{a} = \mathrm{p}_1.\mathrm{p}_2.\ldots.\mathrm{p}_n$$

where a is the new alias, and $\mathrm{p}_1,\ldots,\mathrm{p}_n$ are either folder names or previously defined aliases, subject to the condition that each alias is defined exactly once, and Path declarations are acyclic.

There are three *special aliases*: `Current` refers to the folder of the Package it is used in; `Parent` refers to the parent folder; and `Root` refers to the root folder of the folder hierarchy (thus, the three special aliases correspond to '`.`', '`..`' and '`/`' in Posix systems). Users are discouraged to use the `Root` alias, since it makes reorganising the folder hierarchy difficult; it is mainly intended for usage by tools.

Export and Import. The *local names* are those defined in this package (as opposed to names imported into the package). By default, all local names are exported. Imported names may be re-exported. It is not possible to restrict the export; rather, name clashes and restrictions are resolved on import.

A Package specifies the imported packages in the *ImportPrelude*. The ImportPrelude contains a number of ImportPreludeDecls; each specifies a Package to be imported, plus a number of *import directives*. Import directives allow us to specify:

- Path aliases;
- Local or global import (when importing globally, the imported names are re-exported);
- Qualified or unqualified import (when an import is qualified, the imported Labels for Structural Entities are prefixed with the name of the Package from which they are imported);
- Hiding, revealing, or renaming of imported names (when we hide a name, it is not imported – when we reveal names, only these are imported);

2.4 Attributes

The possibility to define *Attributes* is a central feature for a Structural Entity, cf. Fig. 3. Standard *StructureAttributes* are e.g. the individual *Label* and *Title* of some Section or Unit.

Inheritance of Attributes. Most importantly, *attribute inheritance* to nested Structural Entities relieves the author from specifying Attributes over and over again and avoids cluttering; at the same time, an Attribute may be superseded for a nested "subtree" of Structural Entities.

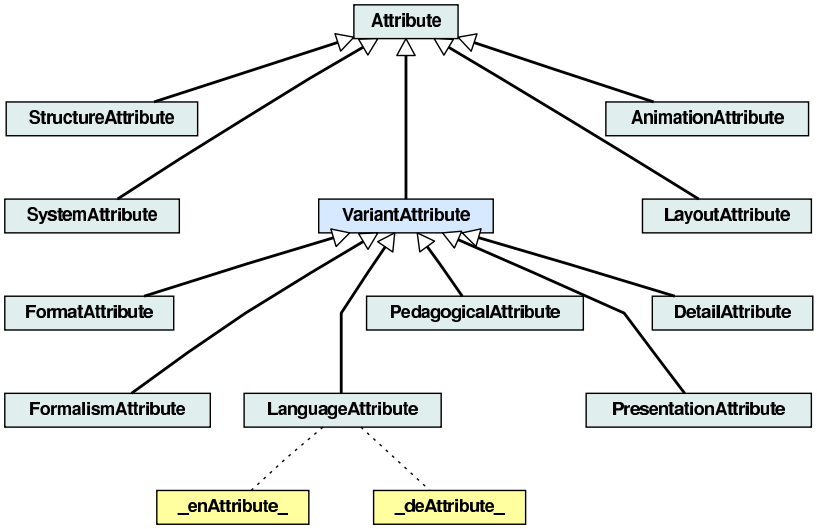

Fig. 3. (Partial) Attribute Ontology

Authors and Version Attributes. Each Structural Entity has an *Authors* and a *Version* Attribute (see also version control in Sect. 6.2). These Attributes record the author(s) of each fragment (inherited to nested Structural Entity). The System automatically keeps track of PriorAuthors and the authorship of individual Revisions.

Layout and Animation Attributes. As will be discussed further in Sect. 7.1 and 7.2, presentation issues such as layout and animation should be separable from the "logical" content of a Structural Entity and should be confined to the necessary only. It is a relief that these can be specified independently as attributes and that attribute inheritance takes care of otherwise tedious repetition of logically irrelevant presentation detail. The specification, for example, that list items on a slide should be rolled out one after another could be specified at the root of a (sub)document and applies to it as a whole unless re-specified for a nested subdocument. Similarly, a revision of such a specification need only be made at its root.

2.5 Variants

Perhaps the most innovative feature of the MMiSS project is the definition of *VariantAttributes* and the management of documents with several different *variants* in a consistent way.

Natural Languages. Let us take the *LanguageAttribute* as an example, cf. Fig. 3. A LanguageAttribute specifies the natural language in which a text is written, following the language codes of IETF RFC 1766 / ISO 639. The default is *en-GB* (British English), overriding the standard *ANY* attribute that is usually the default for the other VariantAttributes. Another example is *de* (German).

Let us now assume that an author wants to manage e.g. English and German documents in parallel. Most probably, the author would want the structure of the two documents to be identical as they are being used for the same purpose, e.g. slides for a Lecture. In this case, s/he may edit two copies of the same document side by side, e.g. in two separate windows of the XEMACS editor (cf. Sec. 5). These two variants should have the same structure, i.e. the same Structural Entities, nested in the same way, where each of them has the same Label as in the other variant, resp. This ensures that the structures of the two variants can be compared, and are consistent and complete, during configuration management (cf. Sec. 6.2). In fact, in the Repository the two variants of the document are merged such that two variants can be identified for each Structural Entity. Thus the author may also edit one variant first and then the other step by step, for each Structural Entity separately, along the structure of the first. Similarly, an individual revision for one Structural Entity is possible, with the two variants side by side.

The structuring relations introduced by these various notions of variants are represented in the MMiSS-system by the variantOf relation.

Format and Formalism Attributes. The *FormatAttribute* takes care of different formats for the same Structural Entity, such as PDF or EPS for Figures, or ASCII, XML or LaTeX for a CASL specification. The *FormalismAttribute* defines the particular Formalism that a Structural Entity (and all its sub-entities) complies with, e.g. a specification language such as CASL; tools may then take advantage of this fact by checking for a special syntax in an ASCII source or generating a LaTeX variant from it for pretty formatting. The Formalism must be related to a particular ontology of FormalismAttributes, cf. also Sect. 3 and Sect. 2.2.

Detail and Presentation Attributes. A document or an individual Structural Entity may exist at several levels of detail during its development, and for different purposes (cf. the *DetailAttribute* in Fig. 3 and Table 1): a set of slides for a Lecture may be refined by adding annotations to LectureNotes, or further to a complete self-contained Course as a hyper-document for self-study. The Contents and Outline denote the underlying structure reflected in the table of contents, and this structure augmented by the various Summaries, resp. At the other end of the scale, conventional articles and books are located.

Table 1 contains another dimension — the *PresentationAttribute* specifies various kinds of presentation media: presentation on Paper, on a (black or white) Board, or as a Hyper document; further kinds specify presentation using an ex-

ternal tool by Replay of a previously conceived script, or by an Interactive presentation with the tool itself.

	Paper	**Board**	**Hyper**
	Text+Pictures	Manual	Hyper-Medium
Contents			
skeleton			
Outline			
abstracts			
Lecture			
presentation	handout	presentation on	laptop browsing
in class	before class	black/white board	during class
Lecture Notes			
annotated	handout	annotated	offline browsing
after presentation	after lecture	manuscript	personal annotation
Course			
self-contained	course script	integrated	personal navigation
for self-study		(manu)script	

Table 1. Detail and Presentation.

2.6 Semantic Interrelation

Declaration of an Ontology. Recall Fig. 1, the example of an ontology of Users and Roles. Such an ontology would be declared in MMiSSLaTeX, the LaTeX extension of MMiSS to represent the Document Constructs (cf. Sect. 5.1), in the OntologyPrelude as follows (partially shown here):

```
\DeclClass{User}{User}{}
 \DeclClass{AcademicUser}{Academic User}{User}
  \DeclClass{StudentUser}{Student}{AcademicUser}
\DeclClass{Role}{Role}{}
 \DeclClass{ReaderRole}{Reader}{Role}
  \DeclClass{TeacherRole}{Teacher}{ReaderRole}
  \DeclClass{StudentRole}{Student}{ReaderRole}
 \DeclClass{WriterRole}{Writer}{Role}
   \DeclClass{AuthorRole}{Author}{WriterRole}
\DeclRel{*-*}{assumesRole}{assumesRole}{}
\RelType{assumesRole}{User}{Role}
```

Consider e.g. `\DeclClass{StudentRole}{Student}{ReaderRole}`, the declaration of a class. The first parameter, `StudentRole`, denotes the particular technical term we use in the ontology; the second, `Student`, the textual phrase that should appear in the text as default (see below); the third, `ReaderRole`,

the superclass of `StudentRole`, from which properties are inherited. Analogously, `\DeclObject{...}` and `\DeclRel{...}` declare objects and relations, resp., whereas `\RelType{assumesRole}{User}{Role}` declares the type of a relation, in this case from the class `User` to the class `Role`; such a declaration may appear several times for different (sub)classes to allow specific typing and "overloading". In the MMiSS ontology, such an *OntologyDecl* operation appears as a *Prelude Operation* in the OntologyPrelude, cf. Fig. 4.

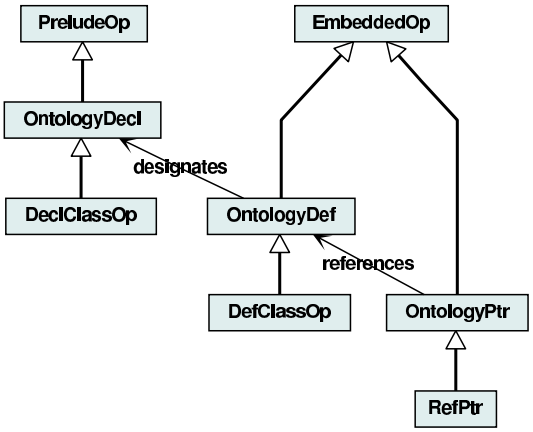

Fig. 4. PointsTo Relations.

Definition and References. An OntologyDecl is a kind of promise, that a corresponding *OntologyDef* will appear somewhere as an *Prelude Operation* in the source text of the document; e.g. `\DefClass{StudentRole}` is the defining occurrence for `StudentRole`, yielding "*Student*" in the formatted document, i.e. the default phrase declared above. Such an OntologyDef operation may appear as an *Embedded Operation* anywhere in the source.

Whenever a class, say, has been declared by an OntologyDecl, the technical term may be referred to simply as `\StudentRole{}` in the source text (or equivalently `\Ref{StudentRole}`), yielding "Student" in the formatted document. If an alternative phrase rather than the default phrase should appear, then e.g. `\Ref[my role as a student]{StudentRole}`, using an optional parameter for this phrase, will yield the desired "my role as a student".

An *OntologyPtr*, e.g. a *Ref*, may appear as an Embedded Operation anywhere in the source, before or after a corresponding OntologyDef. It will yield a hypertext link for the PresentationAttribute Hyper. A full reference may be obtained by `\Reference{StudentRole}`, yielding "Student (see 2.6 , on page 13)".

Note that a relation is predefined as a macro with two parameters, thus `\assumesRole{A }{ B}` yields "A assumesRole B", whereas `\Ref{assumesRole}` yields "assumesRole".

Resolution of Ambiguities. There are at least three reasons for having an extra technical term (the first parameter in an OntologyDecl):

- the default phrase (the second parameter) may be translated into a different language variant of the ontology, assuming that the technical term remains the same for uniformity of language variants,
- the technical term may be renamed upon Import from another package to avoid name clashes while the default phrase remains the same,
- apparent ambiguities may be resolved by having two different technical terms with the same default phrase.

To illustrate the ambiguity issue consider the following example and its source:

a Student assumes the role of a Student

```
a \StudentUser{} assumes the role of a \StudentRole{}
```

An apparent ambiguity (which is usually resolved by context in natural language) is resolved since there are two different technical terms in the example ontology. Note that a hyper-reference references the appropriate OntologyDef correctly.

PointsTo Relations. Consider Fig. 4: a Reference *references* an OntologyDef, an OntologyDef *designates* an OntologyDecl. Both relations belong to the family of *pointsTo* relations, quite similar to the relation family reliesOn.

Inheritance of Relation Properties. Consider an extract of the ontology of relations for Document Constructs:

```
\DeclRel{*-*}{comprises}{comprises}{relatesDocConstructs}
  \DeclRel{<-}{contains}{contains}{comprises}
  \DeclRel{*-*}{includes}{includes}{comprises}
\DeclRel{>}{reliesOn}{reliesOn}{relatesDocConstructs}
  \DeclRel{}{imports}{imports}{reliesOn}
  \DeclRel{}{livesIn}{livesIn}{reliesOn}
  \DeclRel{}{proves}{proves}{reliesOn}
  \DeclRel{}{after}{after}{reliesOn}
\DeclRel{->}{pointsTo}{pointsTo}{relatesDocConstructs}
  \DeclRel{}{designates}{designates}{pointsTo}
  \DeclRel{}{references}{references}{pointsTo}
\DeclRel{->}{variantOf}{variantOf}{relatesDocConstructs}
```

These form a hierarchy, where each relation inherits the properties for its super-relation. Formal properties are indicated by symbols whose semantics is only sketched here: `>` denotes a strict order, `->` denotes an onto-relation, etc.

3 The Content Ontology for MMiSS Courses

In this section we present the variety of courses produced and presented in the MMiSS project (see Sect. 3.1). Moreover we describe the content ontology structure (see Sect. 3.2) and its development process (see Sect. 3.3) that we are using for the MMiSS courses. In general these content ontologies provide the means for establishing the semantic structure to relate different parts of the teaching material. In this sense an ontology is an explicit formal description of concepts in the domain of discourse.

3.1 MMiSS Courses

The MMiSS courses are divided into three areas for differently experienced audiences. Basic courses are provided for the subjects logic, data models and event models. We provide advanced courses for the subjects verification, data specification and reactive systems. Moreover, there exist specialised courses for subjects like formal software development, security and safety critical systems. For each subject shown in Fig. 5 several courses have been prepared and presented at the partner universities of MMiSS and also at other universities such as TU Berlin, TU Dresden, Univ. of Swansea, etc. For a detailed listing of all courses visit the MMiSS website (see [16]).

3.2 The Ontology for Formal Methods

Although ontologies exist for many applications we are not aware of any ontology for formal methods. However, we base our ontology on several approaches for classifying and defining topics related to formal methods such as the ACM classification scheme [1], Astesiano and Reggio's work on defining a schema for formal development techniques [3], Clarke and Wing's survey on formal methods [8] and Steffen's framework for formal methods tools [26].

For describing the ontology of Formal Methods and its instances in UML we use class and object diagrams; cf. the introductory example in Sect. 2.1. The class diagrams serve as representation for the abstract notions such as *Domain*, *Engineering Method*, *Formal Method*, *Formalism*, *Language* and *Tool*. The object diagrams represent the instances of the abstract notions. Typically, particular concepts chosen in a course are represented by object diagrams (see Sect. 3.3).

The most general notion to describe a topic of research or teaching is the notion of Domain (see Fig. 6). A Domain is characterized by a number of *Concept*s and can have zero, one or more subdomains indicated by the association (relation) *isSubDomainOf*. Additionally, a Domain *uses* other Domains (the top level associations like isSubDomainOf and uses are not shown in Fig. 6).

Other classes are specializations of Domain and inherit its associations such as isSubDomainOf and uses. For example, since the class Engineering Method (see Fig. 6) is a specialization of Domain, it inherits the subdomain relation and the relation to Concept. Additionally, an Engineering Method *appliesTo* (zero,) one ore more Domains, it *isSupportedBy* Tools and its pragmatics are described by

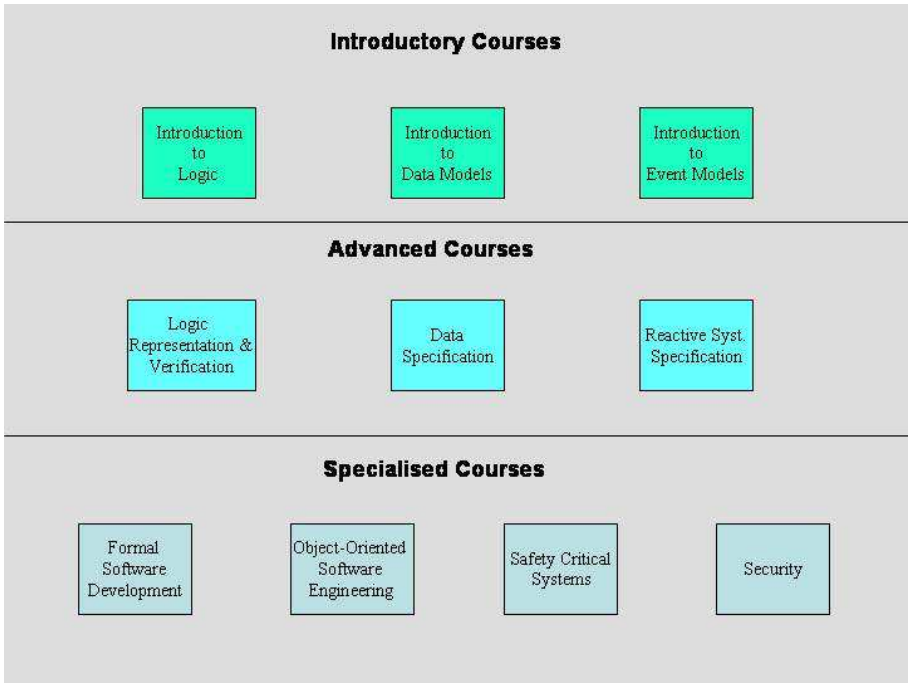

Fig. 5. Structure of MMiSS Courses

Processes (see [3]). Note that in our presentation the multiplicities of an association are indicated below the association name. Moreover, the multiplicities of the association ends are separated by a hyphen.

The class Formal Method (see Fig. 6) is a specialization of Engineering Method with the particular feature that any instance of Formal Method is based on a Formalism. Formal Methods are classified into *Specification Technique* and *Analysis Techniques*; *Verification Techniques* are a subclass of Analysis Techniques (see [8]). Any Specification Technique serves to specify some *System Views* such as the data view, functional behaviour, concurrent behaviour, performance view etc. The class Formalism (see Fig. 7) is another specialization of the class Domain. A Formalism has one or more associated Languages and a Theory consisting of *Definitions* and *Theorems*. Any Language (see Fig. 8) has several *Language Constructs*, *Language Classifications* such as natural, functional, object-oriented or real time language (see [1]), and can be supported by some Tools. Moreover, any Language possesses a *Language Definition* consisting of a *Syntax* and possibly of one or more *Semantics*. We specialize languages into *Programming Language*, *Specification Language*, *Logical Language* and possibly other kinds of Languages which are not represented here.

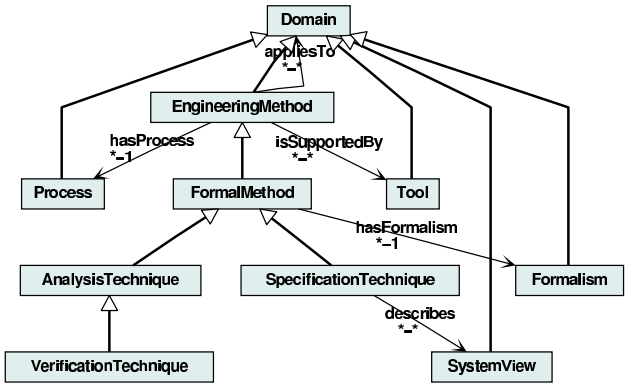

Fig. 6. Semantic Structure of Engineering Method and Formal Method

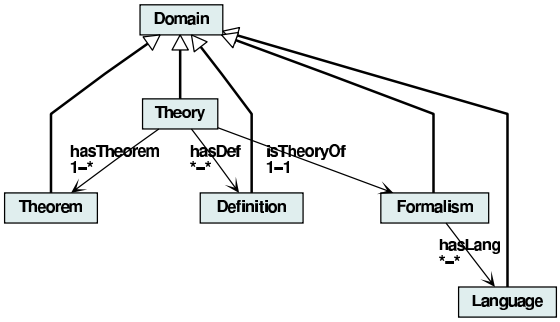

Fig. 7. Semantic Structure of Formalism

3.3 Systematic Construction of Ontologies

For constructing ontologies of particular courses in the area of Formal Methods we proceed as follows: We base the ontology of the course on the general model of Formal Methods as outlined above. In a first step, the general model is extended by new abstract Domains of the course that are not yet covered by the general model. In a second step, object diagrams of the ontology specific to the course are constructed according to the extended general model. We give an example of this procedure by describing (part of) the ontology of the course 'Foundations of System Specification' which is held regularly at LMU München. This course presents formal techniques for specifying and refining complex data

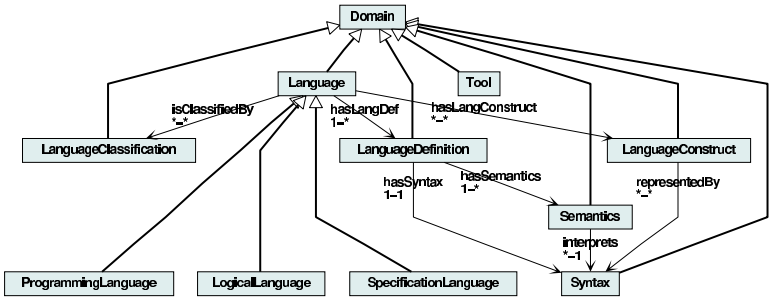

Fig. 8. Semantic Structure of Language

structures, state-based systems and reactive systems. The underlying Formalisms are algebraic specifications based on the Language CASL for data structures; model-oriented specification techniques based on the Language *Z* for state-based systems, and Lamport's Temporal Logic of Actions for reactive systems. In the following we present the ontology for the specification of data structures and state-based systems. In a first step the class diagram of Language is extended by Z and CASL which form two new subclasses of Specification Language (see Fig. 9).

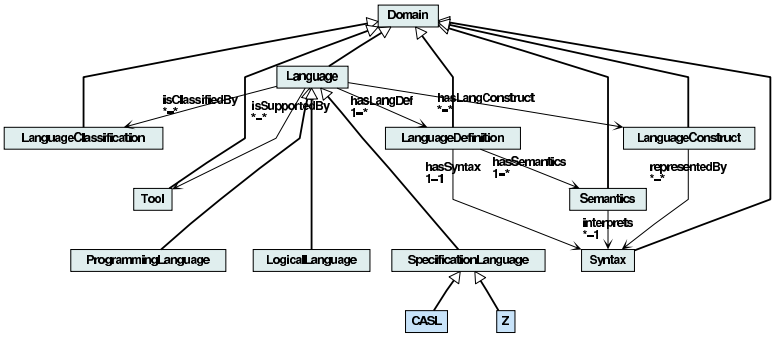

Fig. 9. Extension of the Model by the Languages CASL and Z of the Course

The specific instance of CASL [2,13,6,18] used in the course is the version *CASL 1.0*. It is classified as Specification Language; its Language Constructs are partitioned into *Basic CASL Specification*, *Structured CASL Specification*, *Archi-*

tectural CASL Specification and *Library CASL Specification*. CASL1.0 has formally defined *CASL Syntax* and *CASL Semantics*; its *CASL Toolsuite* consists of parsers, theorem provers and pretty printers (not detailed here; cf. [17]).

The specific instance of Formalism called *FSD Algebraic Specification* of the course uses basic facts about *FSD Signatures*, *FSD First Order Logic* and *FSD Universal Algebra* to explain the associated *FSD Algebraic Specification Theory*. Different notions of refinement including their main properties and a translation from executable specifications to the functional Programming Language SML are presented (see Fig. 10). The instances of the ontology are prefixed by 'FSD' since

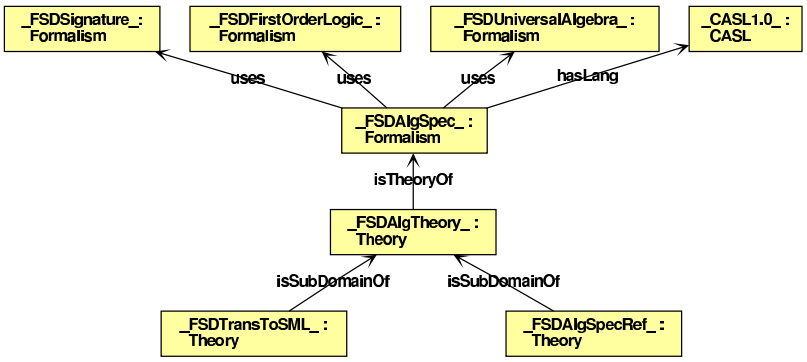

Fig. 10. Ontology for the Alg. Specification Formalism of the Course

they refer to those notions and Theorems of a Formalism of a Theory which are presented in this course. The particular approach of the course to formal development of algebraic specifications is shown in Fig. 10. Algebraic Techniques are used in the context of data specification and the development of functional programs. The chosen development process (the pragmatics) is stepwise refinement.

4 Support Environment

In this section we give an overview on the *Support Environment* integrating the authoring, the management, and the presentation tools for multimedia courses. The Support Environment developed in the MMiSS project aims at the following goals:

Authoring. Authors have to be supported in the development and maintenance of their course material. This comprises the production of new documents, the adaption and revision of existing courses, the import of existing documents

(e.g. slides), but also the composition of existing courses to create a new course. MMiSS supports editing and creating of documents using a synthesis of textual and graphical interaction, combining a graph editor to manipulate the structure of a document and a text editor to edit the actual text.

Management. The development management is responsible for persistency, consistency, and accessibility of the teaching material (Repository), and it has to treat version control, configuration management and change management (Development Manager) for consistency and the dependencies between the document components (via the ontology). It provides an interface mechanism to call external applications to allow for the presentation of such tools during the course or the collection of practical experiences when doing exercises within these tools. Additional support is given by a flexible user management with administration support and the possibility for integrating typical tools for electronic communication.

Presentation The use of the learning material will be supported in different kinds of teaching scenarios: by a Teacher, Tutor or Student on various presentation platforms, or for individualised self-study by a Student using ActiveMath (see Sect. 7.3). Moreover, Students and Correctors use WebAssign for assignments.

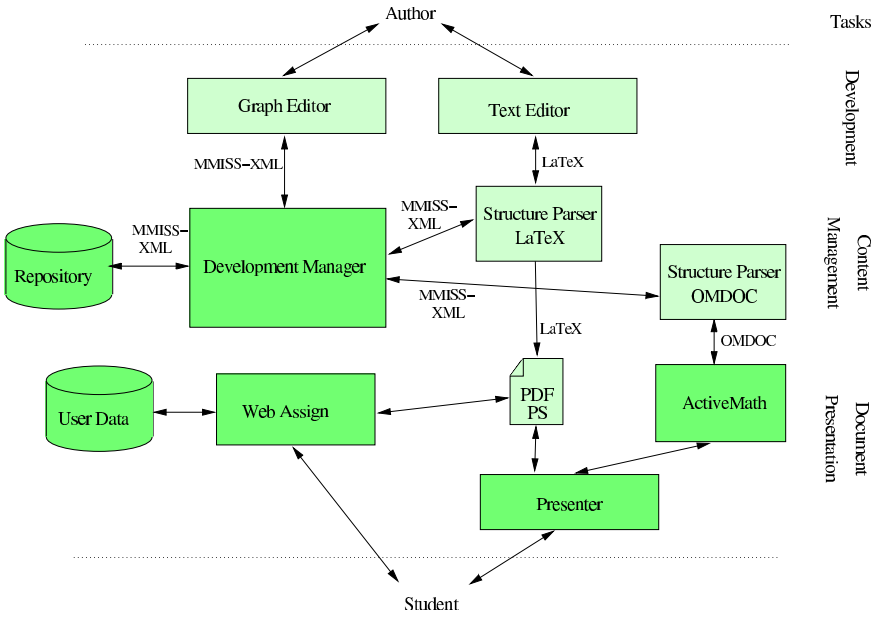

Fig. 11. MMiSS Support Environment System Architecture

To achieve the above goals we have designed an open architecture that integrates subsystems developed by the different partners. Fig. 11 shows the major components of the Support Environment.

In the following sections we illustrate the individual components for authoring, maintenance and presentation in more detail.

5 Authoring Tools

The MMiSS tools allow users to produce, maintain and present course material. As mentioned before, one of the key design ideas is to provide means for making semantic relations in the course material explicit. However, standard languages that are normally used to represent course material, such as PowerPoint or LaTeX, do not support such semantic relations. Therefore, on the one hand MMiSS provides extensions of both languages to cope with such relations, and tools to import existing course material into MMiSS. On the other hand, MMiSS provides an authoring tool to create new course material in a structured way using graph and text editing facilities. Below we illustrate these features in more detail.

5.1 MMiSSLaTeX

LaTeX is a generally accepted language to produce technical documents and course material of high print quality. Although LaTeX comes with some structuring mechanisms it still lacks the expressiveness to formulate many of the structuring mechanisms presented in Sect. 2. In order to support all these mechanisms of MMiSS, we developed a LaTeX-style authoring language called MMiSSLaTeX, consisting essentially of a library of LaTeX class files. MMiSSLaTeX provides commands for each of the structuring operations presented in Sect. 2. Technically, most of them are defined as environments, e.g. for sections, definitions, or lists. The attributes are given as optional arguments to the environments or commands.

A document can be typeset using normal LaTeX with the help of the MMiSSLaTeX class files to generate documents in PDF, Postscript or other formats. For example, Fig. 12 shows a MMiSSLaTeX source text in which the concept of an algebra is defined; with LaTeX this is rendered as the slide shown in Fig. 13. Moreover, MMiSSLaTeX serves as an authoring and input language for the MMiSS repository.

5.2 PowerPoint

In order to migrate existing slides and to relieve the authors from the burden of writing content in an unfamiliar language, tools which allow the import of PowerPoint slides into the MMiSS-repository have been developed. The tool CPoint, developed at the Carnegie Mellon University by Andrea Kohlhase, translates PowerPoint slides into OMDoc, which is another exchange language

```
\begin{Paragraph}[Label=Algebra, Title=Algebra]
  Algebras are models of \Signature{}s.
  \begin{Definition}[Label=DefAlgebra, Title={$\Sigma$-Algebra}]
    An \Emphasis{Algebra} $A= (S_A, \Omega_A)$ for a signature
    $\Sigma=(S, \Omega)$ ($\Sigma$-Algebra) is given by
    \begin{List}[Label=AlgebraComponents, ListType=itemize]
    \item
      for each sort $s\in S$, a \Emphasis{carrier set}
      $A_s\in S_A$;
    \item
      for each operation $\omega:s_1\ldots s_n\rightarrow s$, an
      operation $\omega_A:A_{s1}\ldots A_{sn}\rightarrow A_s$.
    \end{List}
  \end{Definition}
\end{Paragraph}
```

Fig. 12. Example of a MMiSSLaTeX document: Definition of an Algebra

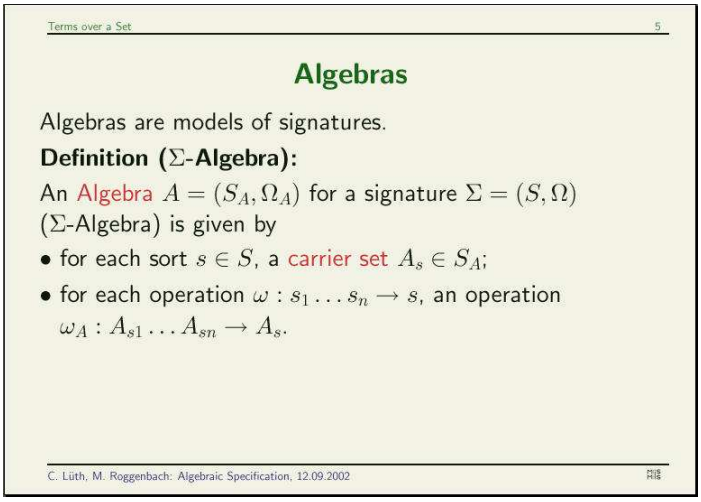

Fig. 13. The example from Fig. 12, rendered as a slide

of the MMiSS-repository; CPoint enriches PowerPoint documents with additional semantical information, like the semantic structuring relations mentioned in Sect. 2, or with other metadata, like for instance authors or date, and finally translates these annotated slides into OMDoc. As OMDoc documents, the slides can be imported into the repository. CPoint makes use of the QMath tool to parse and translate mathematical formulas occurring inside the slides.

5.3 Interactive Course Creation

To edit material, MMiSS provides a graphical interface based on the graph visualisation system daVinci [9,4] and the XEmacs editor [29]. The idea is to visualize and edit the various structuring relations contained in MMiSS documents in a graph editor while a text editor is used to deal with the basic text fragments (like Units, see Sect. 2.2). The predominant interaction paradigm is *direct manipulation* — authors do not have to learn cryptic command lines to interact with the system, they can just point at the entity of interest, or select from a menu of given choices.

5.4 Structure Graph

According to the structure of MMiSS documents in Sect. 2, a Structural Entity is an entity such as Section, Program, Exercise, TextFragment, etc. These entities are related, most obviously by textual nesting, but also by structural Links or References. This structure gives rise to the *structure graph*, which has the various entities as nodes, and the relations as labelled, directed edges. With regard to the comprises relation, the graph is directed and acyclic, but it is not a tree, since a Structural Entity may be included in more than one place (structural sharing).

As an example, consider a real-life lecture series introducing formal program development in the algebraic specification style to undergradute computer science students. One section of this lecture series, corresponding roughly to one lecture, introduces the basic concepts of algebraic specifications. The document is structured as follows:

- it has a short Introduction motivating what is going to come,
- an Abstract summarising the new concepts,
- the main part consisting of Paragraphs, introducing and defining the concepts of Signatures, Algebras, and Terms,
- followed by a short Example (the natural numbers in CASL),
- and closes with a Summary of the new concepts.

Introduction and Summary contain a list enumerating the concepts the user is about to learn, or has just learned. The main parts are Paragraphs, which are structured further: for example, Signatures, Algebra and Terms contain definitions of the corresponding concept. The resulting structure graph is shown in Fig. 14. Note how later Definitions and Examples refer back to earlier Definitions (indicated by dashed arrows), for example to define an Algebra we need to refer to Signatures.

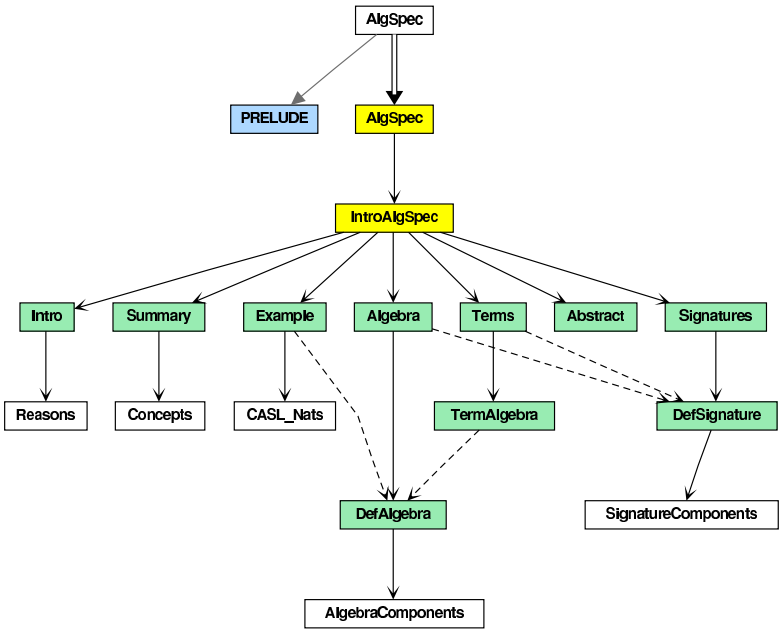

Fig. 14. Structure Graph of Example Document.

Thus, each node represents a Section (shaded yellow), Unit (shaded green), or Atom containing the description of the corresponding notion in more detail. In order to edit this description the user can select a node, corresponding to a Repository object, and its code is loaded into the XEMACS editor (see Fig. 15). A special MMiSSLaTeX mode gives the user additional editing assistance, e.g. to insert environments or commands. Documents can be edited in the MMiSSLaTeX exchange format using a particular MMiSSLaTeX mode; thus other editors may be used as well.

Since a Structural Entity can get quite large (for example, a whole Package), we only display one level in the XEMACS editor; nested Structural Entities are displayed by clicking buttons. For example, Fig. 15 shows the Paragraph labelled ALGEBRA being edited. It contains the definition labelled DEFALGEBRA, which has been opened, and the user is just about to open the list ALGEBRACOMPONENTS.

The Repository objects in the Repository are organised in folders, which allow the grouping of Repository objects much like directories in a file system. Folders may contain other folders, or MMiSS Structural Entities, i.e. Sections, Units or Atoms. The structure graph contains the hierarchy of folders and, at the leaves of

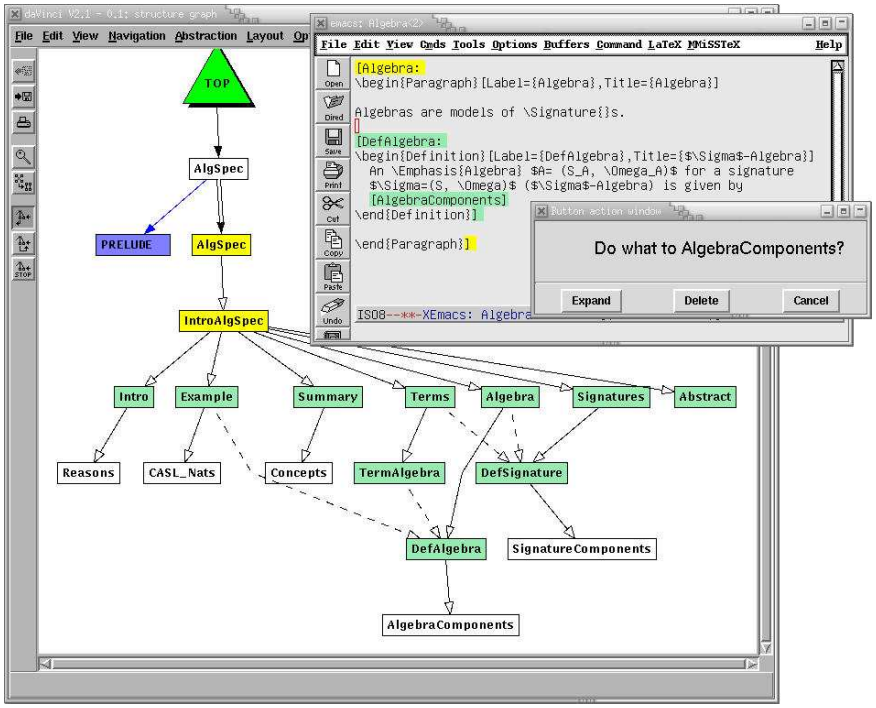

Fig. 15. The Structure Graph: Editing the Definition of an Algebra

this hierarchy, the structure of the Repository objects inside a Package, as nodes, with edges corresponding to the comprises relation (resulting from nesting and structural sharing).

6 Sustainable Development and Maintenance

The MMiSS *Repository* is the central database maintaining MMiSS documents. Sustainable development is supported by fine-grained version control, configuration management and a change management. The structured representation of documents as graphs allows operations to take the structuring into account (see e.g. change management described in Sect. 6.3). It is also the basis for a configuration management to control various versions of a document. We call such graphs together with the possible activities a *development graph*.

The Repository is implemented almost entirely in the functional programming language Haskell [11] in about 60 Kloc. It uses the open source data base BerkeleyDB [25] to store documents. The graph visualisation system daVinci, the graphical user interface library Tcl/Tk [21] and the XEmacs editor are

encapsulated in HASKELL. These encapsulations are available separately, and can be used independently, in particular the Tk encapsulation, called HTk [23].

The content model is generic over the XML DTD used (although of course the structure parsers for the external exchange formats are not), so the Repository can be used for other document formats as well. More importantly, small changes and extensions in the DTD can be implemented directly without needing to recompile the Repository. To parse the DTD and the documents, we use the Haskell XML library HaXML [28].

6.1 Representation

The principal representation and external exchange format for documents in the Repository is MMiSS-XML, a straightforward translation of the Document Constructs introduced in Sect. 2 into XML. However, XML is not meant to be read or written by human users, and tools have their own input formats, hence for presentation and editing purposes, we need external exchange formats. An *external exchange format* is incorporated into MMiSS by implementing a structure parser, which converts documents in the external exchange format, like MMiSS-LaTeX, into MMiSS-XML and back. More external exchange formats will be added if and when editing and presentation tools accepting and requiring these formats shall be incorporated into the MMiSS system.

6.2 Version Control and Configuration Management

The art of keeping track of the evolution of complex systems in general, and complex documents in this particular case, is called *configuration management*. Changes to a documents have to be organised and recorded, such that earlier configurations can always be retrieved. While usually configuration objects are source files, we follow here a *fine-grained* approach using Structural Entities of MMiSS, like Sections, Units or Atoms or associated Attributes, as configuration objects.

The *version graph* is the representation underlying *version control*. Nodes of the graph represent different *versions* of Repository objects while edges denote the *RevisionOf relation*. An author always starts interaction with the Repository by picking a version under development from the version graph. This version will be checked out into the user's local filespace and can then be edited. Fig. 16 (left) shows a typical version graph, as displayed by daVinci. Version 1.5 is the current *working version* (as visualised by the red shade); it is edited as shown in Fig. 15. When finished with editing, a user may *commit* changes back into the Repository (or may just dispose of them silently). New versions of the changed Repository object and of all Repository objects containing the changed Repository object are created. Thus, new versions are propagated upwards: a change in a constituting Repository object results in a new version of the parent Repository object, all the way up to the root folder.

When a Repository object has more than one Repository object which is a revision thereof, we say this is a *span* in the version graph. For example, in

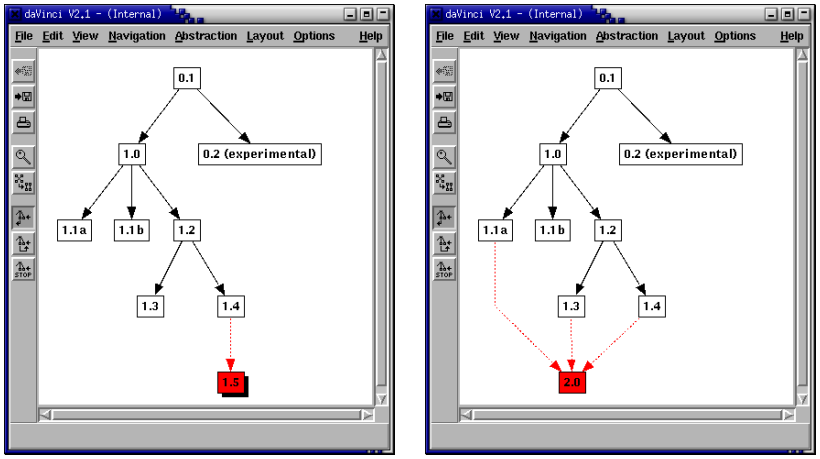

Fig. 16. Version Graph displayed by daVinci (left); Merging of Versions (right).

Fig. 16, there are three spans, starting at the versions 0.1, 1.0 and 1.2. A span corresponds to concurrent revisions of one object. It can be reconciled by a *merge*: users can pick the source versions from which they wish to incorporate all changes into one new version, which then becomes a revision of all the source versions. Fig. 16 (right) shows the merging of versions 1.1a, 1.3 and 1.4 to form a new version 2.0.

6.3 Change Management

The notion of *change management* is used for the maintenance and preservation of consistency and completeness of a development during its evolution. More precisely, we want to have a *consistent configuration* in which all constituents harmonise, versions are compatible, References and Links refer to the proper targets, etc. At the same time, it should be a *complete configuration*: e.g. the promises of forward References and Links should be fulfilled, i.e. they must not be dangling; if we have an English and a German variant of a whole document, then we expect to have a corresponding German variant for each English variant for all constituent Structural Entities, with the same overall structure and relations, and vice-versa.

Such notions are well-known for formal languages; in contrast, natural language used for writing teaching material does not usually possess a well-defined semantics; the notion of consistency is debatable. Different authors may postulate different requirements on the material in order to regard it as being consistent. The existence of an ontology already helps a great deal to check References.

It turns out that the notions of consistency and completeness are closely related to the Document Constructs and relatesDocConstructs relations. For special

FormalismAttributes, additional structuring relations may be explored by special tools operating on these. CASL, for instance, offers the notions of extension, union, etc., to define dependencies between specifications.

The change management keeps track of the various structuring mechanisms described in Sect. 2. Below we will tentatively explore various properties of the individual structuring mechanisms to illustrate possible notions of consistency and completeness and their interaction. Postulating such invariant properties as requirements on the consistency and completeness of a document, and formulating these invariants as formal rules, will enable us to implement a generic and flexible change management that keeps track of the various invariants and informs the user about violations when a previously consistent document has been revised.

Properties of Individual Structuring Mechanisms. For each of the structuring mechanism described above we can formulate various invariants that are prerequisites for consistency or completeness. Some of these are enforced by the underlying structuring language (MMiSSLaTeX) but others may be violated once the user revises a document.

Obviously the comprises relation is reflexive, transitive and antisymmetric denoting an acyclic finite graph (which is actually a subgraph of the structure graph). These properties are trivially enforced by the Document Constructs. We may want to require additional invariants for consistency, e.g. that each major Structural Entity (such as Package or Section) contains at least one Unit or Atom, or that there is at most one Summary in a Section.

Each reliesOn relation or pointsTo relation is irreflexive and acyclic. We would also postulate as a consistency requirement that there is at most one target, i.e. the relations are in fact many-to-one; a completeness requirement is that that there is at least one target, e.g. References must not be dangling; both together require a unique target. Furthermore, for reliesOn relations, we require the target to be presented beforehand. However, the completeness requirement may be weaker for pointsTo relations as we tolerate forward pointers, even to other, future Packages (warnings should be given, though).

Regarding special FormalismAttributes, we adopt their reliesOn relations and corresponding properties. Axioms in CASL, for instance, depend on their global environment resulting from fragments of the Theory that specifies the signature of the symbols used in the Axioms.

The semantics of the variantOf relations depends on the various types of variants. Regarding variants in different languages (or on different levels of detail), we impose the completeness requirement that each variant in one language must have a corresponding variant in the other, for each constituent Structural Entity, with the same overall structure and relations (as an option, for each level of detail, and so on). Similarily one will be able to specify, as a consistency requirement, that all Programs should be in a particular FormalismAttribute, e.g. the programming language HASKELL. A corresponding completeness requirement would be that we have, for each Program, a variant in programming language C and JAVA, e.g. for different Teachers of a course.

Properties of Interactions between Structuring Mechanisms. While the properties mentioned above are specific to an individual structuring mechanism, we will explore possible interactions of different structuring mechanisms and how they can be used to refine consistency and completeness.

Relating the comprises and reliesOn relations (we subsume pointsTo relations here) allows us to formalize constraints regarding the closure of document parts with respect to the reliesOn relation. We may require, for example, that there is a Proof for each Theorem in a Package or that each Reference references an OntologyDef occurrence in this Package unless there is an explicit import. Furthermore, a reliesOn relation between two Structural Entities is propagated along the comprises relation towards the root of the hierarchy of nested Structural Entities. Consider, for example, a Proof in Section A that proves a Theorem in Section B, then Section A reliesOnSection B. Conversely, a reliesOn relation between two Structural Entities cannot be decomposed and propagated towards the leaves. Changing (parts of) one of them can affect the proposed reliesOn relation.

The interaction between the comprises and the variantOf relation is rather subtle and has not fully been investigated yet. For example, we expect the structure of a document with the DetailAttribute Lecture to be a homomorphic projection of the corresponding structure with the DetailAttribute Course.

Similarly, the interaction between the reliesOn relation (or pointsTo relation) and the variantOf relation merits further investigation. It is not clear what kind of relations across variants are desired, if any. In principle, each variant should be closed with respect to reliesOn relations, i.e. all targets should be provided in that variant. An exception might be an explicit pointer to material in a lecture from a course, but then this material should be included in the course anyway as a completeness requirement. The converse is more likely: one might want to make a pointer into a more detailed course or lecture notes document from slides in a lecture.

In any case, the more structure there is, the better are the chances for preserving consistency and completeness; any investment in introducing more reliesOn relations, for example, will pay off eventually. The change management will observe whether revisions by the user will affect these relations and, depending on the user's preferences, emit corresponding warnings. It is crucial to point out that, in contrast to formal developments such as in the MAYA-system [24], there is no rigorous requirement that a document should obey all the rules mentioned above. There may be good reasons, for instance, to present first a "light-weight" introduction to all notions introduced in a Section before giving the detailed definitions. In this particular case, one would want to introduce forward pointers to the definitions rather than making the definitions rely on the introduction; thus the rules are covered. The eventual aim of the MMiSS-design is to allow the user to specify her individual notion of consistency by formulating the rules the relations between the various structuring mechanisms have to obey.

6.4 Foreign Tools and Administration

A User Management component supports a simple user model with different Roles and handles the access rights of Authors, Teachers, Students, Tutors, Correctors, and also ToolDevelopers, SystemDevelopers, and Administrators (cf. Sect. 2.1).

WebAssign. The WebAssign system developed at FernUniversität Hagen (see [7] or http://niobe.fernuni-hagen.de/WebAssign) supports web-based distribution, correction, and administration of course related assignments. Assignments may have interactive parts where system gives direct feedback to the student. WebAssign also manages the integration of external tools (such as compilers) that check student answers or provide help in other ways. In addition, WebAssign provides a flexible administrative support. The WebAssign subsystem is presently being integrated.

7 Presentation

In this section, we concentrate on presentation issues such as layout and animation, and show how they can be realised using the authoring language MMiSSLaTeX.

In general, *presentation* issues should be separated from issues of *re*presentation in an abstract form (MMiSS-XML here), which can also serve as an external exchange format. In fact, the Author should be relieved from tedious formatting as much as possible. Therefore, work is under way to isolate layout and animation as attributes. Ideally, tools will generate different presentation forms automatically. The subsystem ActiveMath, which is developed separately, is integrated via a mapping from MMiSS-XML to OMDoc and provides user-adaptive presentation based on pedagogic rules.

7.1 Layout in MMiSSLaTeX

Annotating Slides. At Universität Freiburg, experiments have been made to enhance slides for the course *Computer-Supported Modelling and Reasoning* (held regularly each year) towards a self-contained online course. We will report on our experiences below; they have led to new insights into the best ways of defining the layout (and animation) of the DetailAttribute LectureNotes and, to some extent, Course.

Usually, slides for a lecture are sketchy and rely on the oral presentation of the Teacher. So in order for the slides to be adequate for self-study, an apppendix has been added to each slide suite containing detailed explanations. There is a rich structure of pointers (hyperlinks resulting from References and Links), mainly going from some item in the lecture slides to an explanation of that item, but also many other pointers to even more detailed explanations, and forward and backward pointers within the slides. For example, whenever there is a sentence starting with "Recall that ...", there is a pointer to the corresponding previous

item in the lecture, usually a Reference to the OntologyDef occurrence of an element declared in the ontology.

In fact, the slides of the lecture have been extended (as a refinement) to a level of detail that we would now regard as lecture notes for review by a Student after attending class; a self-contained online course without any tutoring would require yet a higher degree of verbosity. The tentative experience seems to indicate that different explicit levels of detail, depending on the Student's learning profile, are not so important. The level of detail develops dynamically during a learning session depending on the pointers the Student decides to follow.

However, in its current form the lecture notes is not very suitable for printing. At the least, the detailed explanations would have to be interleaved with the lecture slides so that an explanation immediately follows the slide referred to, and other pointers (References or Links), which are not hyperlinks anymore, have to be augmented by "(see Sect. ...)" or "(see page ...)". This emphasises the need for a more abstract representation format and tools for generating different output formats automatically, cf. Sect. 2.5. In fact, a different presentation for a Reference or Link is now being generated in MMiSSLaTeX, depending on the PresentationAttribute: a hyperlink for Hyper and an extended text "(see Sect. ...)" for Paper.

To give a quantitative assessment of the material involved, one can say that extending lecture slides to a lecture notes at least doubles the size of the sources. A typical lecture notes slide will contain around five pointers. We will further extend the material as Student feedback reveals where more detail is needed.

Board Presentation. The Board PresentationAttribute of MMiSSLaTeX (cf. Sect. 2.5) allows for preparing a 'shooting script' of courses. In addition to the slides to be presented, such a script may also include notes on

- what to write on the board,
- which interactions with the students shall take place,
- important oral remarks etc.

during a lecture. Thus, while the annotated slides for online learning provide help for the students, the Board PresentationAttribute is a means of support to the lecturer during the course presentation. Slides to be presented are included as pictures between text blocks to be written on the board. These text blocks are structured by the same environments as available for slides. During the lecture, this kind of presentation helps to keep overview on the course material: the lecturer sees more than the slide currently presented; personal notes of all kinds can be included; tedious but important things like a uniform numbering of chapters, sections, environments, etc. are done automatically.

While preparing a lecture in the Board style of MMiSSLaTeX, text blocks or graphics can easily be shiftet between slide- and board-presentation thanks to the uniform naming of the structural entities. Technically, this is done by adding or removing the Board PresentationAttribute. This makes it possible to postpone the decision on how to present a certain item to the very last translation before

presentation. In the electronic version of the resulting script, it is possible to run tool demonstrations included in the slides. Thus, one should consider the shooting script prepared in this way as the all inclusive document of a course's presentation.

Concerning the board content, one should be aware that it is of a different type than the material on slides: while slides are intended for presentation to students, only the lecturer will see the contents with the Board Presentation-Attribute. This allows board content to be less detailed, for instance the following might suffice: 'ex-tempore example: model an automaton with the signature provided by the above specification'. From a didactical point of view, such ex-tempore examples — maybe even suggested by the audience — are often better and far more impressive than examples, which are prepared in all details before the presentation. Of course, the same type of argument carries over to proof sketches instead of complete proofs. The Board PresentationAttribute allows also to include this kind of reminders in the course's shooting script.

7.2 Animation in MMiSSLaTeX

With respect to animation, we focus here on a presentation (e.g. of a slide in a lecture, but also of lecture notes in the Hyper variant) where parts are gradually appearing or disappearing in a sequence of displaying steps. The simplest and best-known case is that of an incremental buildup of a page: each step adds new text below the text already presented. These and more complex effects can be very useful in a lecture to illustrate how some complex object is built up step by step; the effect is similar to a presentation on the board. They are even more useful for lecture notes or a self-contained course as no Teacher is available.

So far, such steps have been realised by so-called pause levels in PDFLaTeX using the PPower4 package [10].

Courses involving logic give rise to a particular application of animation effects, namely *animated derivation trees*. A derivation tree is shown in Fig. 17. The LaTeX package proof for drawing such trees has been extended to support animation: the particular logical structure of derivation trees and the general input syntax for such trees is taken into account.

$$\dfrac{\dfrac{[A\vee B]^1 \quad \dfrac{[A]^2}{B\vee A}\,\vee\text{-}IR \quad \dfrac{[B]^2}{B\vee A}\,\vee\text{-}IL}{B\vee A}\,\vee\text{-}E^2}{A\vee B\to B\vee A}\,\to\text{-}I^1$$

Fig. 17: Derivation tree

For each tree, one can specify at which pause levels it should be displayed. For each (sub)tree, one can again specify at which pause levels it should be displayed, *overriding* the specification for the surrounding tree. Derivation trees involve applications of *rules*, e.g. $\to$-*I*. Each rule application can be associated with the *discharging* of an assumption, marked by brackets around the assumption and labelling both the rule and the brackets with a number. We have automated this process: the numbers are administrated using symbolic references (making it easy to compose trees). Moreover, the brackets and their label will by default inherit the pause level from the rule application. For example, one could specify that the whole tree (and hence its root step marked by rule $\to$-*I*) appears from

pause level 4 onwards, whereas the assumption $A \vee B$ at one of the leaves appears from level 2. Then, by default, the *brackets* around $A \vee B$ and the label (here: 1) will appear from level 4 onwards.

Derivation trees can be quite complex and the process of constructing them is very hard to understand based on static illustrations. We therefore found the new style package very useful.

7.3 User Adaptive Presentation in ActiveMath

The ActiveMath project [14] was started independently from and before the MMiSS project and has provided a lot of valuable ideas.

Goals. In the previous sections it became clear that producing on-line learning material involves a lot of effort and that reusability in different contexts and for different presentations and presentation formats is a must in the development of future learning material. As one conclusion, a more abstract, semantical XML knowledge representation, OMDoc [12], has been developed and in addition, presentation tools and other functionalities of the learning environment are strictly separated from the knowledge representation of the learning content in ActiveMath and can thus deliver different output formats, different hyperlinking, different presentations of symbols and formulas, personalized appearances etc.

Apart from these economically and technically-driven developments, a major goal of multimedia on-line learning is a better quality of learning. This objective calls for pedagogically and cognitively motivated features of a learning system which include personalization of content and appearance, the provision of feedback, and presentation according to the learning progress. For instance, a learner becomes bored and less motivated when the material and exercises are too easy for her and not challenging at all. Similarly, the learner's motivation will drop considerably when material and exercises are beyond her capabilities and knowledge mastery level. Therefore, a few advanced intelligent tutoring systems – including ActiveMath – adapt the content and its sequencing to the learners goals, capabilities, and learning preferences/scenario.

Knowledge Representation. ActiveMath was the first system that uses the knowledge representation OMDoc [20]. OMDoc is an extension of the OpenMath `XML`-standard [2]. OpenMath provides a grammar for the representation of mathematical objects and sets of standardized symbols (the content-dictionaries). OMDoc inherits the grammar for mathematical objects from OpenMath and the existing content-dictionaries. In addition, OMDoc defines a framework for the definition of new symbols.

The objectives of OMDoc and MMiSS-XML are quite similar: OMDoc was originally more tailored towards mathematical content and is being extended

[2] http://www.openmath.org

now; MMiSS-XML has had more general objectives, is more tailored towards the document Document Constructs described above and the input language MMiSS-LaTeX; MMiSS-XML can be mapped to OMDoc and vice-versa — efforts are presently being made for further unification.

The metadata in core-OMDoc include the Dublin Core [27] metadata such as `contributor` and `publisher`. The ActiveMath DTD extends OMDoc (see e.g. [15]) and contains additional – pedagogically motivated – metadata such as difficulty or field of an exercise and the prerequisite-of relation of instructional items for a concept that allow even more customization of the document delivery to the student and her learning situation.

Adaptive Presentations. Thanks to a user model that stores and updates the learner's preferences, goals, activities, and mastery levels, ActiveMath is able to present the learning material in a user-adaptive manner, content-wise and presentation-wise. In the table-of-contents a color-annotation informs the student about her mastery level for concepts to be learned.

The flexibility of the presentation process also chooses a low slide-verbosity or a high script-verbosity of the material according to the learner's needs. A slide presentation can automatically be (hyper-)linked to the more verbose explanations and other instructional items from the script sources.

Mathematical objects/symbols in the presentation have a semantic annotation that points to the meaning of the symbol in the content dictionary. This enables functionalities such as copy and paste of mathematical formulas to a service system's console.

The transformation from assembled XML content items to the actual output format is realized with a modular presentation process with style sheet application at its heart. Currently, ActiveMath can realize LaTeX PDF output formats that are well-suited for printing as well as HTML output format augmented by MathML-presentation for mathematical symbols that is well-suited for browser presentation output.

Learning-Effective Features in ActiveMath. ActiveMath offers several other features that are known to improve learning. In particular, it has a generic mechanism for integrating service systems/tools for active and exploratory learning, such as computer algebra systems or tools for formal software development.

A dictionary can be called from the material or by explicit search in the dictionary. It displays the definition of a concept and, if required, also the concepts and instructional items that are somehow related to that concept, e.g., examples illustrating the concept or exercises training the concept.

The learner can resume studying where she left off last time. She can manipulate (rename, delete) those (listed) materials she has studied previously. A notes facility enables the learner to take personal or group notes corresponding to items in the learning material. The user model is open and inspectable.

ActiveMath is customisable to teacher's and learner's needs and easily configurable to pedagogical strategies and knowledge resouces.

8 Conclusion

In this paper we have presented the methodology, the techniques and tools of the MMiSS-project to support multimedia instructions in safe and secure systems. Summing up, the developed infrastructure allows a user

- to develop transparencies, lecture notes, complete courses
- to work on the board, with transparencies, interactively with tools
- to embed mathematical formulae, programs, etc.
- to manage e.g. English and German variants in parallel
- to publish complete and consistent packages
- to (partially) re-use the transparencies of a colleague
- to be made aware of the changes made by colleagues
- to develop a uniform terminology among various authors, and
- to have support for sustainable development.

Experiences. The system has been gradually introduced, over the duration of the project, into the normal teaching activities of the project partners. For example, the two semester course TeCS (Techniques for the development of Correct Software) at Universität Bremen provides a gentle introduction to formal methods for software development. It deals with sequential as well as with reactive systems, using the algebraic specification language CASL [2,13,6,18] and the process algebra CSP, e.g. [22], resp. On the tool's side, the theorem prover Isabelle and the model checker FDR play central roles. Besides simple exercises explaining single concepts, the TeCS problem sheets also include more complex tasks like specifying a family game (Nine Men's Morris) in CASL; verifying a simple interpreter within Isabelle/HOL; modelling a file system in CASL at both the requirements and the design level; proving the refinement relation between these two specifications in HOL/CASL.

Presenting TeCS using the presentational part of MMiSSLaTeX has been quite successful. For the *author*, the overhead to produce course material within the MMiSSLaTeX format is negligable compared to other presentation systems. Besides the usual benefits of a computer based presentation like 'no slide confusion', the MMiSSLaTeX integration of tool demonstrations in the slides encourages the *teacher* to enliven the lectures by live demonstrations on the computer. The *students* are fond of the readability, the consistent markup, and the download-friendly PDF-filesize of the slides. It should be mentioned that these positive results also arise from a cautious usage of computer based presentations: about half of the course material has been taught in 'classical style' using a blackboard. A poll among the students of TeCS gave the result that this is an optimal mixture.

State of the Project and Future Developments. The project has made good progress during its first two years. Many lectures have been converted to the initial LaTeX-oriented input format, with good quality output as slides in

PDF-format. This material is now awaiting further coordination and refinement, as well as semantic interlinking via an ontology and using development graphs in the repository. The Development Manager, and other editing and authoring tools, have been made available in a first version.

While the project has achieved a satisfactory, consistent state, a lot still has to be done: the documentation has to be improved, various bits and pieces have to be completed (e.g. layout and animation attributes), etc. We hope that the planned extension mechanisms will facilitate future developments considerably.

MMiSSForum. As the open source model is used, teaching materials and tools are freely available to achieve a much wider national and international take-up. To assist this, a MMiSSForum is has been set up with German, international, and industrial members, to evaluate the emerging curriculum and assist its development and distribution; you are welcome to join ([16]). The Advisory Board advises the project from a scientific as well as an industrial perspective, with a view to future applications. To go with the planned deployment at universities, a number of well-known German companies have already, through the various industrial contacts of the project partners, expressed an interest in measures for further in-house training.

Support and Partners. The MMiSS project is being supported by the German Ministry for Research and Education, bmb+f, in its programme *"New Media in Education"* from 2001 to 2004. The project partners are

- Universität Bremen (Krieg-Brückner, Drouineaud, Eckert (now at Darmstadt), Gogolla, Kreowski, Lindow, Lüth, Mahnke, Mossakowski, Peleska, Roggenbach (now at Swansea), Russell, Schlingloff (now at HU Berlin), Schröder, Shi)
- FernUniversität (Distance Education University) Hagen (Poetzsch-Heffter (now at Kaiserslautern), Bealu, Kraemer, Sun, Jelitto)
- Universität Freiburg (Basin (now at ETH Zürich), Klaedtke, Smaus,Wolff),
- Ludwig-Maximilians-Universität München (Wirsing, Kröger, Knapp, Hennicker, Meier, Zhang),
- Universität des Saarlandes (Hutter, Melis, Autexier, Siekmann, Stephan, Goguadze, Libbrecht, Ullrich).

Acknowledgement We are grateful to the members of the Advisory Board, M. Kohlhase (Carnegie-Mellon University, Pittsburgh), V. Lotz (Siemens AG, München), H. Reichel (Technische Universität Dresden), W. Reisig (Humboldt Universität Berlin), D.T. Sannella (University of Edinburgh), and M. Ullmann (BSI [Federal Institute for Security in Information Technology], Bonn), for their advice.

References

1. Computing Classification System [1998 Version]. http://www.acm.org/class/.
2. E. Astesiano, M. Bidoit, B. Krieg-Brückner, H. Kirchner, P. D. Mosses, D. Sannella, and A. Tarlecki. CASL – the common algebraic specification language. *Theoretical Computer Science*, 286:153–196, 2002.
3. E. Astesiano and G. Reggio. Formalism and method. In *Theoretical Computer Science*, pages 236(1–2), 2000.
4. b-novative GmbH. davinci presenter web site. http://www.b-novative.com/products/daVinci/.
5. K. Baclawski, M. K. Kokar, P. A. Kogut, L. Hart, J. Smith, W. S. Holmes III, J. Letkowski, and M. L. Aronson. Extending UML to support ontology engineering for the semantic Web. *Lecture Notes in Computer Science*, 2185:342–360, 2001.
6. H. Baumeister, M. Cerioli, A. Haxthausen, T. Mossakowski, P.D. Mosses, D. Sannella, and A. Tarlecki. CASL semantics. In P.D. Mosses, editor, CASL *Reference Manual*. [19], Part III.
7. J. Brunsmann, A. Homrighausen, H.-W. Six, and J. Voss. Assignments in a virtual university - the webassign-system. In *Proceedings of the 19th World Conference on Open Learning and Distance Education*, Vienna/Austria, June 1999.
8. E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. In *ACM Computing Surveys 28*, pages 626–643, 1996.
9. M. Fröhlich. *Inkrementelles Graphlayout im Visualisierungssystem daVinci*. PhD thesis, Dissertation, Universität Bremen, 1998.
10. K. Guntermann and C. Spannagel. *PPower4 Manual*. TU Darmstadt, 2002.
11. Haskell web site. http://www.haskell.org/.
12. M. Kohlhase. OMDOC: Towards an internet standard for mathematical knowledge. In E. R. Lozano, editor, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2000*, LNAI. Springer Verlag, 2001. See also http://www.mathweb.org/omdoc.
13. CoFILanguage Design Group, B. Krieg-Brückner and P.D. Mosses (eds.). CASL summary. In P.D. Mosses, editor, CASL *Reference Manual*. [19], Part I.
14. E. Melis, E. Andres, G. Goguadse, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: System description, 2001.
15. E. Melis and C. Ullrich G. Goguadse, P. Libbrecht. Wissensmodellierung und -nutzung in ACTIVEMATH. *KI*, to appear(1):12–18, 2003.
16. MMiSS web site. http://www.mmiss.de.
17. T. Mossakowski. CASL: From semantics to tools. In S. Graf and M. Schwartzbach, editors, *TACAS 2000*, volume 1785 of *Lecture Notes in Computer Science*, pages 93–108. Springer-Verlag, 2000.
18. P. D. Mosses and M. Bidoit. CASL — *the common algebraic specification language: User Manual*. Lecture Notes in Computer Science. Springer. To appear.
19. P. D. Mosses (ed.). CASL — *the common algebraic specification language: Reference Manual*. Lecture Notes in Computer Science. Springer. To appear.
20. OmDoc. http://www.openmath.org.
21. J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994.
22. A.W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1998.
23. G. Russel and C. Lüth. Htk — graphical user interfaces for haskell programs. http://www.informatik.uni-bremen.de/htk/.

24. S.Autexier, D.Hutter, T.Mossakowski, and A.Schairer. The development graph manager MAYA (system description). In H. Kirchner and C. Reingeissen, editors, *Algebraic Methodology and Software Technology, 2002*, volume 2422 of *Lecture Notes in Computer Science*, pages 495–502. Springer-Verlag, 2002.
25. Sleepycat Software. Berkeley DB. `http://www.sleepycat.com/`.
26. B. Steffen, T. Margaria, and V. Braun. The electronic tool integration platform: Concepts and design. In *International Journal on Software Tools for Technology Transfer (STTT) 1*, pages 9–30, 1997.
27. The Dublin Core Metadata Initiative. Dublin core metadata initiative - home page, 1998. `http://purl.org/DC/`.
28. M. Wallace and C. Runciman. Haskell and XML: Generic combinators or type-based translation? In *International Conference on Functional Programming ICFP'99*, pages 148– 159. ACM Press, 1999.
29. Xemacs web site. `http://www.xemacs.org/`.