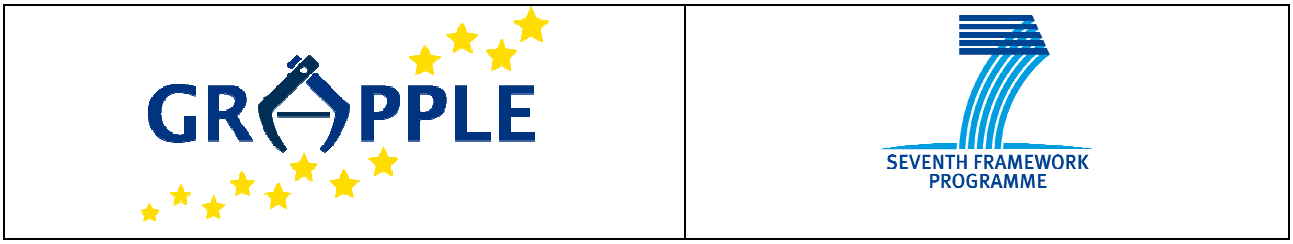

GRAPPLE

D3.3c Version: 1.0

User Guide on the Concept Adaptation Model Tool

Document Type	Deliverable
Editor(s):	Fawaz Ghali (Warwick)
Author(s):	Fawaz Ghali (Warwick), Jonathan Foss (Warwick)
Reviewer(s):	Patrick Pekczynski (IMC), Daniel Krause (LUH)
Work Package:	WP3
Due Date:	July 2010
Delivery Date:	2010-07-08
Version:	1.0
Version Date:	2010-07-08
Total number of pages:	25

Abstract: This deliverable outlines the current implementation of the Conceptual Adaptation Model (CAM) tool. The CAM tool is a tool for creating CAM instances with the help of the DM tool and CRT tool (see deliverables D3.1a and D3.2a). The CAM will be system-independent and will be a model for straightforward, graphical authoring of Adaptive Hypermedia. The CAM tool has already been described in deliverable D3.3a. In this document, we give an update on the design of the technical specifications for the CAM format and tool as well as for the Shell, integrating the tools into the Grapple Authoring tool (GAT). Finally a user guide is included, to help authors create and modify CAMs.

Keyword list: authoring tool, concept adaptation model, CAM tool, domain model, conceptual adaptation model, concept relationship type

Summary

The objective of the *CAM model* is to be a *system-independent high-level model for straightforward, intuitive graphical authoring* of Adaptive Hypermedia. The objective of the *CAM tool* is to be a tool that illustrates the CAM model, thus allowing for system-independent, high-level modelling for straightforward, intuitive graphical authoring of Adaptive Hypermedia. The CAM tool was previously described in deliverable D3.3a and is a tool for creating CAM instances with the help of the DM tool (as described in deliverable D3.1b) and the CRT tool (described in deliverable D3.2b).

This deliverable outlines the initial implementation of the Conceptual Adaptation Model (CAM) tool. The CAM tool is a tool for creating CAM instances with the help of the DM tool and CRT tool (see deliverables D3.1a and D3.2a). The CAM tool has already been described in deliverable D3.3a. In this document, an update is given on the design of the technical specifications for the CAM format and tool as well as for the Shell, integrating the tools into the Grapple Authoring tool (GAT). Finally a user guide is included, to help content authors create and modify CAMs.

A prototype is made available online at http://dyn070.win.tue.nl:8080/GAT/

Authors

Person	Email	Partner code
Fawaz Ghali	f.ghali@warwick.ac.uk	Warwick
Jonathan Foss	j.g.k.foss@warwick.ac.uk	Warwick

Table of Contents

Table of Figures

List of Acronyms and Abbreviations

CAM	Conceptual Adaptation Model; also called: Adaptive Story Line; Adaptation Strategy
CRT	Concept Relationship Type(s); also called: (Pedagogical) Relationship Type(s)
DM	Domain Model
GAL	GRAPPLE Adaptation Language
GAT	GRAPPLE Authoring Tool
GALE	GRAPPLE Adaptive Learning Environment
GUMF	GRAPPLE User Model Framework
GRAPPLE	Generic Responsive Adaptive Personalised Learning Environment
GUI	Graphical User Interface
LAG	Layers of Adaptation Granularity
UM	User Model
UUID	Universally Unique Identifier

Note:

In order to provide clearer terminology to users, the phrases CAM, CRT and DM have been renamed in GAT to *Course*, *Pedagogical Relation* and *Domain* respectively.

1 Introduction

The objective of the *CAM model* is to be a *system-independent high-level model for straightforward, intuitive graphical authoring* of Adaptive Hypermedia. The objective of the CAM tool is to illustrate the CAM model, with its specified attributes. The CAM tool is a tool for creating CAM instances with the help of the DM tool (as described in deliverable D3.1a and D3.1b) and the CRT tool (described in deliverable D3.2a and D3.2b).

The main deliverable for the task T3.3b is the integration of the CAM tool into the GRAPPLE Authoring Tool (GAT). This document is meant to provide an update upon the design of the CAM tool and the Shell that integrates the DM, CRT and CAM tool into the Grapple Authoring Tool (GAT).

The CAM model is a system-independent high-level model. The CAM model will support straightforward, intuitive graphical authoring of Adaptive Hypermedia. The CAM tool has already been described in deliverable D3.3a. In this document, an update is given on the design of the technical specifications for the CAM format and tool as well as for the Shell, integrating the tools into GAT. Finally a user guide is included, to help content author create and modify CAMs.

Work package 3 is mainly connected to WP5, WP9, WP10, WP7 and WP8.

WP5 explores the suitability of existing standards for expressing the various models developed in GRAPPLE. The interaction between WP5 and WP3 results in an understanding of to what extent the CAM model can be expressed in various inputs, roles, interactions, adaptation methods & techniques within the limits of the explored standards. Specifically D5.3a, b and c perform the matching to the available LMS and IMS-LD.

WP7 is concerned with the integration of all different components into a coherent learning environment. The GAT has to communicate with the GALE (Grapple Adaptive Learning Environment), delivery environment for deploying courses as well as the GUMF (Grapple User Modelling Framework). Hence there is a direct link between this deliverable and D7.1 (a and b) about *the operational infrastructure specification and design* and D7.2 (a, b and c) about *Data models and interoperability*.

WP9 is mainly concerned with gathering requirements, providing documentation for and evaluating the experience in higher education. WP10 performs a similar role, with a focus on industrial outreach. WP3 and this deliverable in particular are related to D9.1, *Requirements Specification by Academic Users on Integrated Adaptive Learning Services*, *First Documentation and Training for GRAPPLE Users*. This deliverable is also related to the evaluations performed in WP8, specifically D8.2 *Evaluation Guidelines*.

The rest of the document is structured as follows: Section 2 provides an outline of the CAM model, section 3 explains the XML format that is used to describe the CAM, and in particular describes the changes that have been made to the format since deliverable D3.3b. Section 4 describes the implementation of the CAM tool, with section 5 providing a user guide to the CAM tool. A conclusion is presented in section 6.

2 Summary of the CAM model

This section revisits the design of the CAM model as it has been described in deliverable D3.3a for the benefit of the reader. The CAM model is based upon lessons learnt from the AHAM [13] and LAOS [5] models, as well as being inspired by the multi-model, metadata-driven approach to content adaptation [6], and has incorporated ideas from other models, such as Dexter [8], XAHM [2], Munich [10], UWE [9] and ADAPT [7]. The section is organised as follows. First shortly short introduction of the relevant parts of the AHAM and LAOS models. Then the specific requirements of the CAM model that are not already in AHAM or LAOS will be investigated and finally it will be indicated how the different layers in the CAM model interrelate.

2.1 AHAM

The AHAM [13] reference model for Adaptive Hypermedia Systems (AHS) describes adaptive applications as consisting of three main layers:

- **The Domain Model (DM)** describes concepts, groups them in a hierarchical structure and defines arbitrary concept relationships, possibly of a special domain concept relationship type (domain CRT). In principle, a DM can be “imported” from a simple subject domain ontology, except for the concept relationships that have a pedagogical meaning.
- **The User Model (UM)** also defines concepts but with user specific attributes, e.g., knowledge level, learning style, preferences etc. Typically the UM is an overlay model of the DM, meaning that for every concept in DM there is a corresponding concept in the UM.

- **The Adaptation Model (AM)** defines the adaptation behaviour. It typically consists of condition-action rules or event-condition-action rules that define how user actions are transformed into UM updates and into the generation of presentation specifications. There are two types of rules:

o generic adaptation rules are connected to CRTs; this for instance allows to define a knowledge update rule for visiting pages and a prerequisite rule for determining the suitability of concepts; depending on whether all prerequisites are satisfied; an author only has to specify concept relationships and an authoring tool can then generate the corresponding adaptation rules automatically;

o specific adaptation rules apply to specific concepts of the domain model and can be used for a very rare adaptation rule or for defining an exception to a generic adaptation rule; authoring such specific adaptation rules requires knowledge of the language in which such rules are defined (and which is system-dependent).

In the AHA! system [6], a graphical authoring tool, the Graph Author, is used to define the DM and draw a graph of concept relationships (of different types). As the name “graph” suggests, concept relationships are (unary or) binary, whilst in AHAM there is no restriction to the number of concepts that together may form a relationship.

2.2 LAOS

The LAOS model [5] is a five-layered extension of AHAM, that responds to and improves upon some of the limitations of AHAM, with special focus on authoring and reuse issues. In LAOS, different aspects of the adaptation model are distributed over multiple layers in the model [10], in particular the:

- **domain model (DM)**, similar to the AHAM model DM, with the exception of allowing only domain-specific information (concepts and links), unlike in AHAM;
- **goal and constraints model (GM)**, extracting and concentrating all goal-related (e.g., pedagogical, for educational applications) information previously intermingled with domain information in the AHAM model;
- **presentation model (PM)**, extracting and concentrating all presentation information previously intermingled with domain information in AHAM.
- **user model (UM)** similar to the one in AHAM.
- **adaptation model (AM)**, extending the ideas of genericity and specificity, using:
    - generic adaptation rules: instead of connected to a type of event (such as visit), allowing for specification of the combination of type of event, type of concept, type of relation and determining, based on this ternary combination, the UM or PM updates. Such rules can be applied to any domain map.
    - specific adaptation rules: similar to the generic rules, but applied to a specific concept, identified from a given domain map (this is mainly used for backward compatibility with previous types of adaptive hypermedia, like AHAM, where concepts were connected only in this explicit, specific way, and only reusable with one DM).

This way, pedagogical information, e.g., can be expressed in the GM alone, and kept separate from other information. The PM also describes the final look and feel of the presentation as well as the quality of service parameters (e.g. for mobile devices).

CRT’s in LAOS are also of different types, depending on the model they belong to:

- e.g., the domain CRT’s only describe domain relations (such as part-of, IS-A or relatedness relations as available in the MOT authoring tool [3] built on LAOS) similarly
- CRT’s in the GM describe only goal-related relations (pedagogic relations for the educational domain, such as AND-OR relations with pedagogical labels, as available in the MOT).

The adaptation model allows for different levels (and thus degrees) of reuse of adaptation, by conforming to the LAG framework [4], [11]: establishing as a first level the (event-)-condition-action rules. As in AHAM, they are the building stones for adaptation, and also forming the assembly-language type of adaptation specifications. At the second level, allowing for more sophisticated adaptation languages [3] (such as LAG [4], [11] or LAG-XLS [4], [11]). Finally, at the highest level, adaptation strategies, comprising reusable, annotated storylines of adaptation and personalisation, that can be applied to various domain models.

This type of layered structure allows reuse of each layer separately, beside the type of reuse described above, and is therefore more flexible and more appropriate for authoring, where the 'write-once use-many' paradigm is most relevant.

2.3 CAM

In the GRAPPLE project, the authoring framework used is even more general and flexible: it contains an extensible number of layers, which may be different for each application. This is due to the fact that for a specific application, page-presentation and quality-of-service parameters might need to be stored independently if they are major components of that application, whilst for another one, the approach of storing them together as in LAOS is acceptable. Application-dependent new layers should be allowed to be defined by authors. However, the DM, UM and AM layers are described as mandatory, as without these basic functionalities no adaptive system will work (DM for the underlying domain information, UM to describe the user and the AM to describe the adaptation rules).

This leads to the following list of general requirements (GR) for the GRAPPLE toolset:

GR1: The GRAPPLE authoring tool should allow an extensible number of layers, however, a DM, UM and AM layer are mandatory.

GR2: The GRAPPLE authoring tool should be visual.

GR3: The GRAPPLE tool should contain a UM tool/service (matching the mandatory UM layer of GR1).

GR4: The GRAPPLE authoring tool should contain a DM authoring tool/service (matching the mandatory DM layer of GR1).

GR5: The authoring tool should contain an AM authoring tool (matching the mandatory AM layer of GR1).

The combination of GR2 and GR5, as well as the prior experience with the AHA! graphical authoring tool led to the conclusion that adaptation authoring should deal separately with CRT types and their application. That is why GR5 is subdivided into:

GR5.1: The GRAPPLE authoring tool should contain a CRT type tool.

GR5.2: The GRAPPLE authoring tool should contain an adaptation strategy (or story line) creation tool, called CAM (conceptual adaptation model).

There will always be a DM and UM layer and at least one layer with adaptation aspects. Therefore, the structure of GRAPPLE authoring is a generalisation of the AHAM model, and either equivalent with, or a generalisation of the more refined LAOS model. The GRAPPLE authoring shell tool will comprise a DM authoring tool, UM authoring tool, a CRT authoring tool and a CAM authoring tool. The presence of the CRT ensures that the complexity of authoring specification is hidden in the definition of these concept relationships, whilst the presence of the CAM ensures a high-level, non-programming access to authoring behaviour specification. In this paper, due to its important role in integrating all the other authoring elements, as well as due to the lack of space, the focus is on the CAM model and tool.

Because of the limited nature and specificity of the DM and CRT authoring and the general requirement GR1, this leads to the following requirements (R) for the CAM authoring tool:

R1. The CAM should allow for an extensible number of layers.

R2. DM concepts and relations representation should keep the same look and feel between different layers (DM, CAM layers).

R3. CRT relationships and placeholders should keep the same look-and-feel between the different layers (CRT and CAM layers).

As the CAM tool inherits the requirements from GR2, as well as is influenced by R2, R3, the requirement R4 can be defined as follows:

R4. The CAM tool should be able to visually represent the different layers (e.g., showing only one type of relationship or showing multiple relationships, but each with a different representation - in the simplest case, colour).

The relationships defined in the different CAM layers do not yet express the actual adaptation that will take place. A prerequisite may be translated to a rule that will change the presentation of links to concepts, but it may also be translated to the conditional inclusion of a prerequisite explanation (fragment). The translation of CAM structures to actual adaptation rules is described and exemplified in Section 3.

2.4 Relations between the layers of the GRAPPLE authoring tool

The Domain Model (DM) describes the domain concepts, their attributes and their semantic relationships. These relationships do not attach any adaptive behaviour but merely indicate a semantic link. In the CAM tool, concepts from the DM tool can be selected and the adaptive behaviour can be attached to these concepts. For a concept to appear in the final adaptive course, it needs to be selected in the CAM. This implies the following requirement:

R5. There should be a common shell for the DM and the CAM tools, and a drag & drop facility from the former to the latter, to allow for one (or more) domain concepts to be selected and dragged into the CAM tool. **Note:** the Google Web Toolkit (GWT)[1] was chosen for this common shell.

The CAM model only puts together the final picture of the adaptive behaviour, much like the pieces in a jig-saw. The types of possible adaptive behaviour are defined in the CRT library created by the CRT tool. The CAM can link a number of domain concepts using a certain CRT, and therefore concretely establish the adaptive behaviour defined in the CRT in a more generic way. CRTs dragged into the CAM tool can then be populated with domain concepts from the DM tool, as long as the particular CRT allows for that particular type of concept.

2.5 Relations between the CAM and the adaptation engine(s)

The CAM model groups domain descriptions and adaptation descriptions into adaptation strategies or adaptive story lines. These story lines need to be exported to an adaptation engine. This engine can be the GRAPPLE adaptation engine developed in the GRAPPLE project or any other engine that can work with (or has converters) for one or more of the exported adaptation languages. The CAM tool will export several languages of various levels: the CAM XML language, a combination of CAM and CRT language, the adaptation engine's own Grapple Adaptation Language (GAL), possibly also other languages, such as the LAG language [4], [11].

3 Specification of the CAM Format

In order to exchange CAMs with other GRAPPLE components a data format has to be defined which can be written by the CAM Tool and read by other components. Most importantly CAMs have to be read by converters to adaptive display engines such as GALE (see deliverable D1.3c).

CAMs are expressed in XML format and follow an XML schema specification which defines the structure of the XML format.

In this section the common explanation of the CAM format and structure is given, followed by the XML schema definition. Finally this section outlines which changes have been made to the CAM definition since the last deliverable D3.3b. A complete example of a CAM definition can be found in Appendix **Error! Reference source not found.** of deliverable D3.3b.

3.1 Structure of the CAM format

For the reason of compatibility with the other formats of the GRAPPLE authoring tools, a common wrapper format has been defined. Another benefit is that the web service can treat the different models transparently the same, without any knowledge of the differences between the types of model (DM, CRT, CAM), and can use the header info for quick look-up of models.

3.2 Common header

The following XML code outlines the common structure of GRAPPLE authoring tool models. Each model has a common header part, where the metadata of the model are located. Metadata include title, common description, creation- and update time, author, authorisation and the UUID of the model. Below is a short outline of the common parts of the XML format.

```
<model>
  <header> <!--the header is shared between all models-->
    <modeluuid>660e8400-e29b-41d4-a716-446655440000</modeluuid>
       <!--the modelUuid is a unique identifier for the model.-->
```

[1] http://code.google.com/webtoolkit/

```
    <modeltype>DM</modeltype>
       <!--Types can be DM, CRT, CAM or their VR and Simulation equivalent-->
    <authoruuid>660e8400-e29b- a716-41d4-446655440000</authoruuid>
       <!--the author UUID is the unique identifier of the author, wo created the model-->
    <authorisation>readwrite</authorisation>
       <!-- authorisation for all other author; the original author always has all permissions.
       permissions can be none (no access by other authors) read (read-only access) and
       readwrite (read and write access) -->
    <creationtime>1242904243000</creationtime> <!-- the timestamp of creation of the model -->
    <updatetime>1242904243000</updatetime> <!--the timestamp the model was last updated -->
    <title>sun-example-dm</title> <!--the title of the model-->
    <description>sun-example-dm</description> <!--the description of the model-->
  </header>
  <body><!-- the body contains the actual content of the model -->
    <cam> <!-- depending on the model there will be a CAM, CRT or dm tag to indicate the model
type-->
       <!--content of the model (in this case cam) see below for more details →
    </cam>
  </body>
</model>
```

3.3 The CAM formats

This section describes the CAM languages as they were implemented in the first prototype of the GAT. What has been updated compared to the preliminary version in deliverable D3.3a will also be indicated.

3.3.1.1 CAM visual language

The visual representation, which is meant for authors to create adaptive story lines, expresses the CAM instance in a visual way and is the only language that is intended for the non-programmer author to work with. This language is essential for ensuring the lowest possible threshold in authoring of adaptation.

The CAM languages are all based upon the well known concept of graphs. Hence, the main components are nodes and links. More specifically the nodes are sockets, placeholders that can contain sets of concepts from a DM instance and the links are the relationships from the CRT instance. There is an additional grouping operator as well.

To summarise there are the following components that make up the graphical representation:

- A socket, a placeholder for sets of DM Concepts, with default representation:

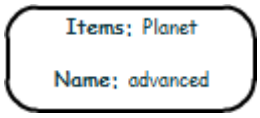

Figure 1. A socket with one concept

- CRT instances, with an image based representation and a number of anchor hook locations, where sockets can be hooked into. The default representation is:

Figure 2. Example of prerequisite CRT representation in CAM visual language

- Groups (visually shown as sockets) of concepts of placeholders (groups can have anchors too, not shown here). The default representation is:

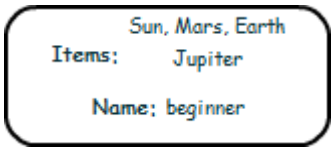

Figure 3. Grouping of concepts

All of these defaults can be replaced by images included from a library, as well as by simple descriptors, as defined by the CAM internal language.

3.4 The CAM visual language

There are a number of updates on the CAM visual language. The main update is that the notion of sockets has been introduced. This notion is similar to what has been envisioned as *'grouping of concepts'* in D3.3a. Also the notion of placeholder concepts has been abandoned to an extent. The format does not allow for concepts that are not attached to sockets. As such, there is no representation for placeholder concepts. The sockets act as the placeholders that can contain concepts. The final update is the cosmetic update of the representation of CRTs, concepts and sockets.

3.5 CAM internal language

The CAM internal language is the main format for storage and manipulation of the CAM tool. It is designed to be used by other systems, whilst they interface with the CAM tool. Prior experience with the LAG language [4] shows that non-programmer authors prefer not to be at all involved at the level as described by this language. On the other hand, programmers or authors with a Computer Science background prefer the more 'hands-on' experience. For the latter authors only, the CAM XML language could be potentially used directly to describe adaptive behaviour. Prior research also showed that an XML-based language is preferable, as it is both more portable and perceived as easier to manipulate than a pure programming language, as in the development of the LAG-XLS language [4]. The LAG-XLS language was only aimed at learning styles, and in order to adopt a wider scope of adaptivity, a new language has to be created.

The CAM format can be used to describe, not only direct CRT relations, but also more global information and flow of a course:

- Start and end states can be described with a special CRT. The CRT only differs in the GAL [12] code simply stating "start" or "end".
- Main concepts of a course, useful for visualising the course, can be indicated in the name/metadata of the sockets.
- Learning goals and objectives can be derived from the selection of start and end states and main concepts.

Below the new CAM language is illustrated by an example. This type of CAM output is seen as XML descriptions of groups of concepts and *named typed* relations between them. For example, if concept A is to be seen before concept B, a CAM would be:

```
<model>
  <header> .. </header>
<body>
  <cam>
    <camInternal>
      <domainModel>..</domainModel>
      <crtModel> .. </crtModel>
         <crt>
           <uuid>cf5de7f5-12b5-4720-92e5-736cac59985b</uuid>
           <shape>diamond</shape>
           <colour>#C0C0C0</colour>
           <camSocket>
             <uuid>e9b45bd0-6013-11de-8a39-0800200c9a66</uuid>
             <socketId>cf5de7f5-12b5-4720-92e5-zzzzzzzzz</socketId>
             <position><x>100</x><y>250</y></position>
             <size>10</size>
             <shape>rectangle</shape>
             <colour>#006633</colour>
             <entity><dmId>201-de-8a39-0800200c9a66</dmId></entity>
           </camSocket>
           <camSocket>
             <caption>target</caption>
             <uuid>f539bae0-6013-11de-8a39-0800200c9a66</uuid>
             <socketId>2b5-4720-92e5-pppppppppppp</socketId>
             <position><x>100</x><y>400</y></position>
             <size>10</size>
             <shape>rectangle</shape>
```

```
            <colour>#006633</colour>
            <entity><dmId>11de-8a39-0800200c9a66</dmId></entity>
          </camSocket>
        </crt>
    </camInternal>
  </cam>
</body>
</model>
```

The CAM language consists of two parts the *header* and the *body*, the header is the common header described above. The *body* part contains the information specific for the CAM model. The body part contains the following information:

- The domainModels and crtModels in use, as described in sections 5.1 and 5.2
- A number of *crt* tags, containing the instantiation of CRTs from *the crtModel* with actual concepts from *the domainModel*.

The *crt* tag contains the following information:

- A unique identifier, the uuid
- The shape and colour of the CRT, for displaying the CRT in the CAM tool(this is optional), defaults will be used if the tags are omitted.
- An optional position element. The position of a CRT needs to be given if the CRT is not binary (1 or >2 sockets). For binary CRTs the position is calculated, based on the position of the sockets.
- A number (≥1) of CAM sockets (camSockets).

The *camSocket*s contain the instantiation with the actual concepts, they contain the following information:

- An optional caption, a name for the socket, for the authors' assistance.
- A unique identifier the uuid
- A socket id (socketId), the unique id of the specific socket in the crtModel
- A position, the position where the socket should be displayed in the CAM Tool.
- An optional shape and colour, the shape and colour a CAM socket should have; a default will be used if the tags are omitted.
- A number of entity tags.

The entity contains the instantiation with the actual concepts, they contain the following information:

- Zero or one dmID element, containing the ID of the concept in the domain model.
- Zero or more labels for the resource of a concept and an optional location for a resource
- A number of relationshipType elements, where a requirement for a concept to be involved in a certain relationship in the DM model can be expressed (e.g., the entity should have participated in an IS-A relation)
- The relative position in the socket instance, if it differs from the default.
- The size of the entity in the socket, if it differs from the default.
- The shape of the entity in the CAM instance, if it differs from the default.
- The image of the entity in the CAM instance, if it differs from the default.
- The colour of the entity in the CAM instance, if it differs from the default

The description of the subject domain and behaviour semantics of the CRT called 'prerequisite' needs to be separately imported from the DM and CRT repositories. These descriptions are then included in the *domainModel* and *crtModel* parts.

3.6 XML Schema of CAM internal language

This section contains the XML schema specification of the CAM XML format. CAMs which are created by the CAM tool are valid against this schema specification. This specification is supposed to be used by all GRAPPLE tools and components which read CAMs. An example of a CAM which implements this specification can be found in Appendix **Error! Reference source not found.**.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/CAM-
external-oct2009" xmlns:tns="http://www.example.org/CAM-external-oct2009"
elementFormDefault="qualified" xmlns:Q1="http://www.imsglobal.org/xsd/imsvdex_v1p0"
xmlns:Q2="http://grapple-project.org/GAT/" xmlns:Q3="http://www.grapple-project.org">

    <import schemaLocation="dm-vdex-extensions.xsd" namespace="http://www.grapple-
project.org"></import>
    <import schemaLocation="CRToct2009.xsd" namespace="http://grapple-project.org/GAT/"></import>
    <import schemaLocation="vdex.xsd"
namespace="http://www.imsglobal.org/xsd/imsvdex_v1p0"></import>
    <element name="model" type="tns:modelType"></element>

    <complexType name="modelType">
       <sequence maxOccurs="1" minOccurs="1">
                <element name="header" type="tns:headerType" maxOccurs="1"
minOccurs="1"></element>
                <element name="body" type="tns:camBodyType" maxOccurs="1" minOccurs="1"></element>
       </sequence>
    </complexType>

    <complexType name="headerType">
       <sequence>
                <element name="modeluuid" type="tns:UUIDtype" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="modeltype" type="string" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="authoruuid" type="tns:UUIDtype" maxOccurs="1"
minOccurs="1"></element>
                <element name="authorisation" type="tns:authorisationType" maxOccurs="1"
minOccurs="1"></element>
                <element name="creationtime" type="dateTime" maxOccurs="1"
minOccurs="1"></element>
                <element name="updatetime" type="dateTime" maxOccurs="1" minOccurs="1"></element>
                <element name="title" type="string" maxOccurs="1" minOccurs="1"></element>
                <element name="description" type="string" maxOccurs="1" minOccurs="1"></element>
       </sequence>
    </complexType>

    <complexType name="camBodyType">
       <sequence>
                <element name="cam" type="tns:camType" maxOccurs="1" minOccurs="1"></element>
       </sequence>
    </complexType>


    <simpleType name="authorisationType">
       <restriction base="string">
                <enumeration value="read"></enumeration>
                <enumeration value="write"></enumeration>
                <enumeration value="readwrite"></enumeration>
       </restriction>
    </simpleType>

    <simpleType name="UUIDtype">
       <restriction base="string">
                <pattern
                         value="[a-z0-9]{8}\-[a-z0-9]{4}\-[a-z0-9]{4}\-[a-z0-9]{4}\-[a-z0-9]{12}">
                </pattern>
       </restriction>
    </simpleType>


    <complexType name="camType">
       <sequence>
                <element name="camInternal" type="tns:camInternalType"></element>
       </sequence>
    </complexType>
```

```
    <complexType name="camInternalType">
       <sequence>
                <element name="domainModel" type="tns:dmModelType" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="crtModel" type="tns:crtModelType" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="crt" type="string" maxOccurs="unbounded" minOccurs="0"></element>
       </sequence>
    </complexType>


    <complexType name="positionType">
       <sequence>
                <element name="x" type="int"></element>
                <element name="y" type="int"></element>
       </sequence>
    </complexType>

    <complexType name="camSocketType">
       <sequence>
                <element name="caption" type="string" maxOccurs="1"
                         minOccurs="0">
                </element>
                <element name="uuid" type="tns:UUIDtype" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="socketId" type="tns:UUIDtype" maxOccurs="1"
                         minOccurs="1">
                </element>
                <element name="position" type="tns:positionType"
                         maxOccurs="1" minOccurs="1">
                </element>
                <element name="shape" type="string" maxOccurs="1" minOccurs="0"></element>
                <element name="colour" type="string" maxOccurs="1" minOccurs="0"></element>
                <element name="entity " type="tns:entityType" maxOccurs="unbounded"
minOccurs="0"></element>
       </sequence>
    </complexType>

    <complexType name="entityType">
       <sequence>
                <element name="dmID" type="tns:UUIDtype" maxOccurs="1"
                         minOccurs="0">
                </element>
                <element name="label" type="string" maxOccurs="unbounded"
                         minOccurs="0">
                </element>
                <element name="relationshipType " type="string" maxOccurs="1"
minOccurs="0"></element>
                <element name="position" type="tns:positionType" maxOccurs="1"
minOccurs="0"></element>
                <element name="size" type="int" maxOccurs="1" minOccurs="0"></element>
                <element name="shape" type="string" maxOccurs="1" minOccurs="0"></element>
                <element name="image" type="string" maxOccurs="1" minOccurs="0"></element>
                <element name="colour" type="string" maxOccurs="1" minOccurs="0"></element>
       </sequence>
    </complexType>

    <complexType name="dmModelType">
       <sequence>
                <element name="model" type="tns:dmModelType2" maxOccurs="unbounded"
minOccurs="0"></element>
       </sequence>
    </complexType>

    <complexType name="dmModelType2">
       <sequence>
                <element name="header" type="tns:headerType"></element>
                <element name="body" type="tns:dmType"></element>
       </sequence>
    </complexType>

    <complexType name="dmType">
       <sequence>
                <element name="vdex" type="Q1:vdexType" maxOccurs="1" minOccurs="1"></element>
       </sequence>
```

```
      </complexType>

      <complexType name="crtModelType">
         <sequence>
                  <element name="model" type="tns:crtModelType2" maxOccurs="unbounded"
  minOccurs="0"></element>
         </sequence>
      </complexType>

      <complexType name="crtModelType2">
         <sequence>
                  <element name="header" type="tns:headerType" maxOccurs="1"
                           minOccurs="1">
                  </element>
                  <element name="body" type="tns:crtType" maxOccurs="1"
  minOccurs="1"><complexType></complexType></element>
         </sequence>
      </complexType>

      <complexType name="crtType">
         <sequence>
                  <element name="crtdialect" type="Q2:crtdialecttype"
                           maxOccurs="1" minOccurs="1">
                  </element>
                  <element name="comment" type="Q3:contentType" maxOccurs="1"
                           minOccurs="1">
                  </element>
                  <element name="visualrepresentation" type="string"></element>
                  <element name="crtsockets" type="string"></element>
                  <element name="adaptationbehaviour" type="string"></element>
                  <element name="constraints" type="string"></element>
                  <element name="associateddmrelations" type="string"></element>
         </sequence>
      </complexType>
  </schema>
```

3.7 CAM external language

The CAM external language is the portable format for the output for the Grapple Adaptation Tool (GAT). In essence it is a scaled down version of the CAM-internal language, with only the information relevant for delivery. Therefore it does not have any information for visualising the CAM model but it does have all other information.

Below the new CAM External language is illustrated by the same example used in section 3.5. This type of CAM output is seen as XML descriptions of groups of concepts and *named typed* relations between them. For example, if concept A is to be seen before concept B, a CAM external would be:

```
  <model>
    <header> .. </header>
  <body>
    <cam>
      <camInternal>
        <domainModel>..</domainModel>
        <crtModel> .. </crtModel>
           <crt>
             <uuid>cf5de7f5-12b5-4720-92e5-736cac59985b</uuid>
             <camSocket>
               <uuid>e9b45bd0-6013-11de-8a39-0800200c9a66</uuid>
               <socketId>cf5de7f5-12b5-4720-92e5-zzzzzzzzz</socketId>
               <entity><dmId>201-de-8a39-0800200c9a66</dmId></entity>
             </camSocket>
             <camSocket>
               <caption>target</caption>
               <uuid>f539bae0-6013-11de-8a39-0800200c9a66</uuid>
               <socketId>2b5-4720-92e5-pppppppppppp</socketId>
               <entity><dmId>11de-8a39-0800200c9a66</dmId></entity>
             </camSocket>
          </crt>
      </camInternal>
    </cam>
  </body>
</model>
```

4 Implementation and Integration of the CAM Tool

The basic design of the CAM tool has already been outlined in D3.3a. It has been described that the tool has a user interface where the author can create, edit and modify CAMs. CAMs should be stored on the server side using a web service and a database behind the web service.

Because of a more efficient implementation the common functionalities of DM tool, CRT tool, and CAM tool have been integrated into the GRAPPLE Authoring Tool (GAT) shell. The functionalities of the GAT shell are provided to all of these tools. The most important functionalities are providing a container including a menu bar and handling the creation and updating of data models.

4.1 The GAT shell and Web Service

The different models (DM, CRT and CAM) each have an authoring tool to author the specific models. The goal of GAT is ultimately to make it possible for authors (teachers) to create adaptive courses. In order to present a coherent approach and not to frustrate the authors with an array of different tools it is very important that GAT offers the tools under one umbrella. During the implementation it has been decided to take this one step further and to create one completely integrated tool. For an author there is now only one tool, GAT, which offers DM, CRT and CAM editing functionalities. The fact that the models and tools have been developed in parallel by different teams is completely transparent to the user.

A guide for tool implementers for integrating their tool with the GAT shell can be found in Appendix **Error! Reference source not found.** of deliverable D3.3b. The shell uses the *dockable* Flex library[2] and the *birdeye ravis* library[3]. The main functionality it offers is a fully functional toolbar. Implementing tools are expected to extend *ClosableVBox* (from the dockable library) and react to events.

4.2 Implementation of the CAM Tool

The CAM tool was implemented, as designed in D3.3a and D3.3b. The CAM tool architecture has a lot of similarities with the DM- and CRT- tool architectures as specified in D3.1a and D3.2a, as well as according to requirements R02 and R03. The CAM tool is a web-based application schematically shown in the figure below.

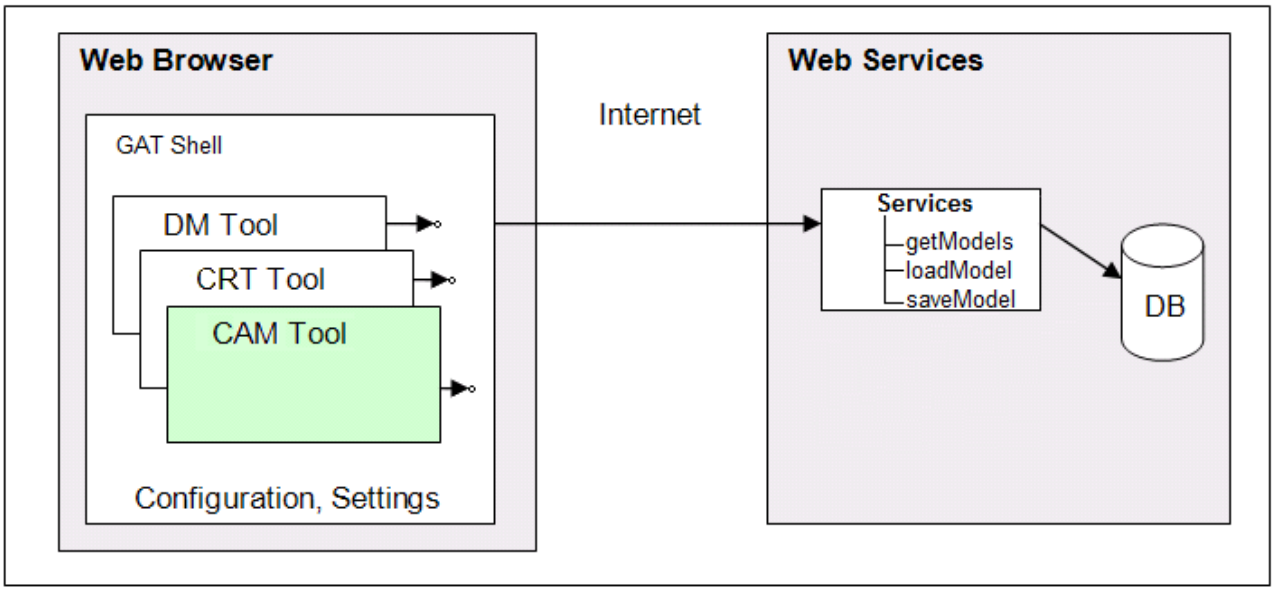

Figure 4. GAT Tool Architecture

The CAM tool has two components as shown follows:

1. The client part, running in the author's browser, authoring a CAM.

[2] http://www.goozo.net/dockableflex/docs/

[3] http://code.google.com/p/birdeye/

This will be a graphical tool, encapsulated in the general GRAPPLE Authoring tool using the shell described in the next section, using the provided common functionality such as configuration and passing information between the different tools (CAM, CRT, DM). Each of the tools can be seen as a separate component of the GRAPPLE authoring tool.

2. The server partis running as a web service on a web server available for access over the internet. Its main task is storing and retrieving models (CAMs, DMs and CRTs) a library. It is worth noting that there is only one web service handling the different models transparently. For this reason, all models have a common header, see section 3.2. The CAM tool, the DM tool and CRT tool use the generic functionalities in the GAT shell to communicate with the web service.

4.3 Integration of the CAM Tool with the GAT shell and Web Service

The GAT shell is an Adobe Flex[4] library which can be integrated into the tools. It provides a graphical container, a menu and the functionality for handling models. Figure 5 depicts how the CAM Tool is integrated with the GAT shell.

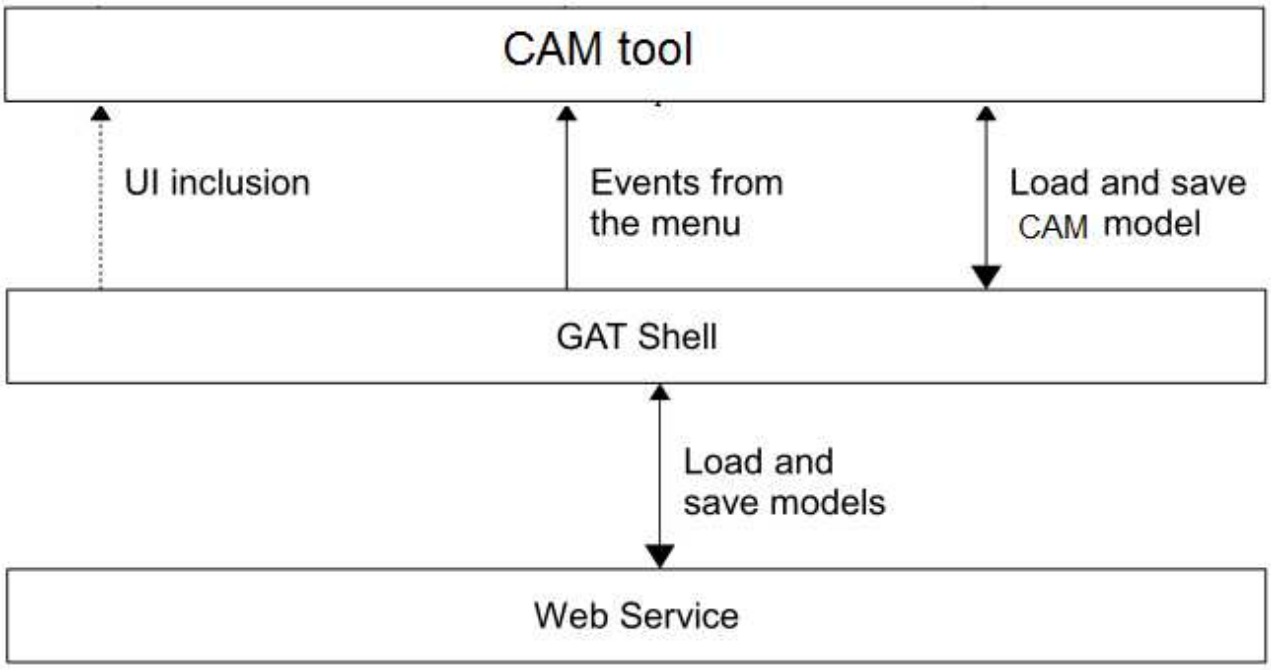

Figure 5: The CAM Tool integrated in the GAT shell

The GAT shell provides the menu and when a user activates a menu, items have to be sent to the CRT Tool. The most important events are "open model", "save model", and "save as". These events are received by the application logic which activates the transformation of the internal CRT representation to and from XML format. The XML code will be sent to or retrieved from the GAT shell. The GAT shell is responsible for saving and loading the models to and from the web service.

5 User Guide on the CAM Tool

The user guide for the CAM tool is part of the user guide for the integrated GAT. The user guide will be made available to the users via the tool itself. Users can select "*Help->Help*" from the menu, which will open a HTML page containing the user guide. The user guide is also part of deliverable D9.2 and will be updated based upon opinions from WP9 collaborators as well as the results of the formal review of deliverable D9.2. As the user guide is integrated into the tool, evaluation may show results regarding its usabilitybut there is no specific evaluation of the user guide planned. In the scope of WP9, some preliminary tests in the GRAPPLE tutorial at the ECTEL conference in 2009 in Nice have taken place, in order to highlight some of the more prominent bugs and issues.

[4] http://www.adobe.com/products/flex/

The CAM will consist of all domain models and CRT descriptions used and their instantiations. The author can create a new CAM by clicking the “Create a New Course” button on the welcome screen, as shown in Figure 7.

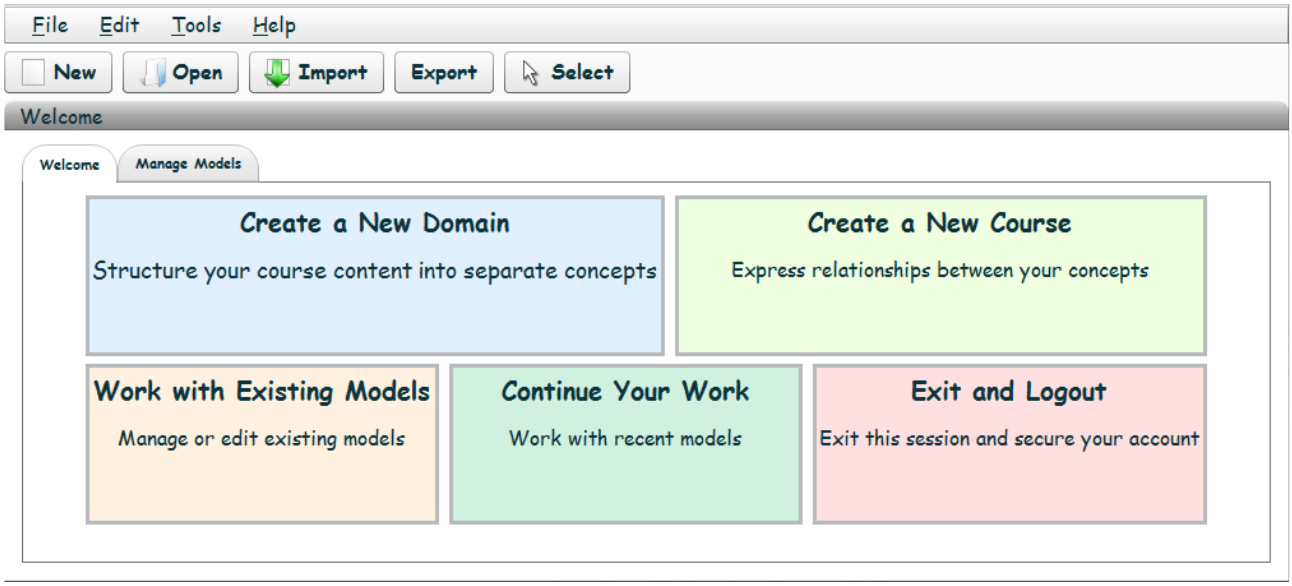

Figure 6. Opening the GAT tool

Alternatively, the author can create a new CAM by selecting “*File->New->Course”* from the menu bar as depicted in Figure 7:

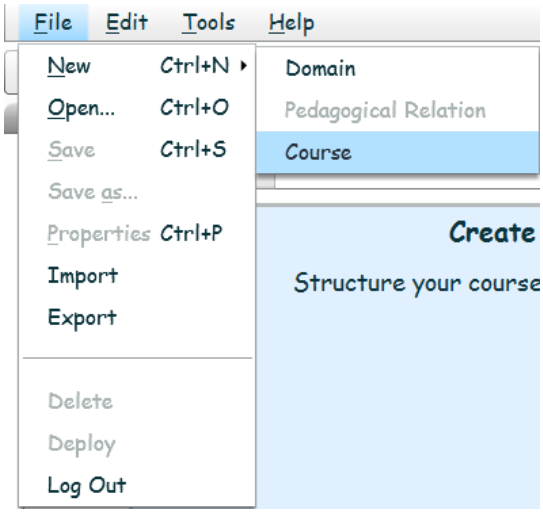

Figure 7. Creating a new CAM

The author has to name the new CAM and the CAM model can be created, by pointing on the CAM editor and right-click. As shown in Figure 8, a new CRT instance can be inserted.

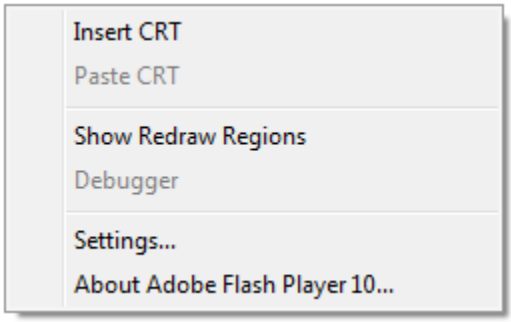

Figure 8. Inserting a CRT instance into a CAM

5.1 Inserting CRTs

A dialog is presented to the author and a CRT model (s)he wishes to insert an instance of. See Figure 9.

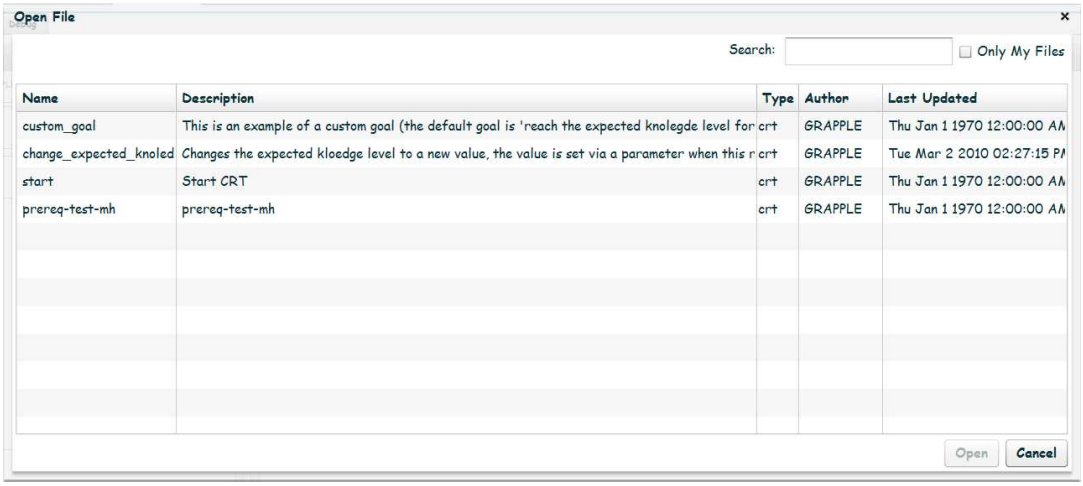

Figure 9. Choosing a CRT to insert

Figure 10 shows the drag and drop facility to insert a new CRT

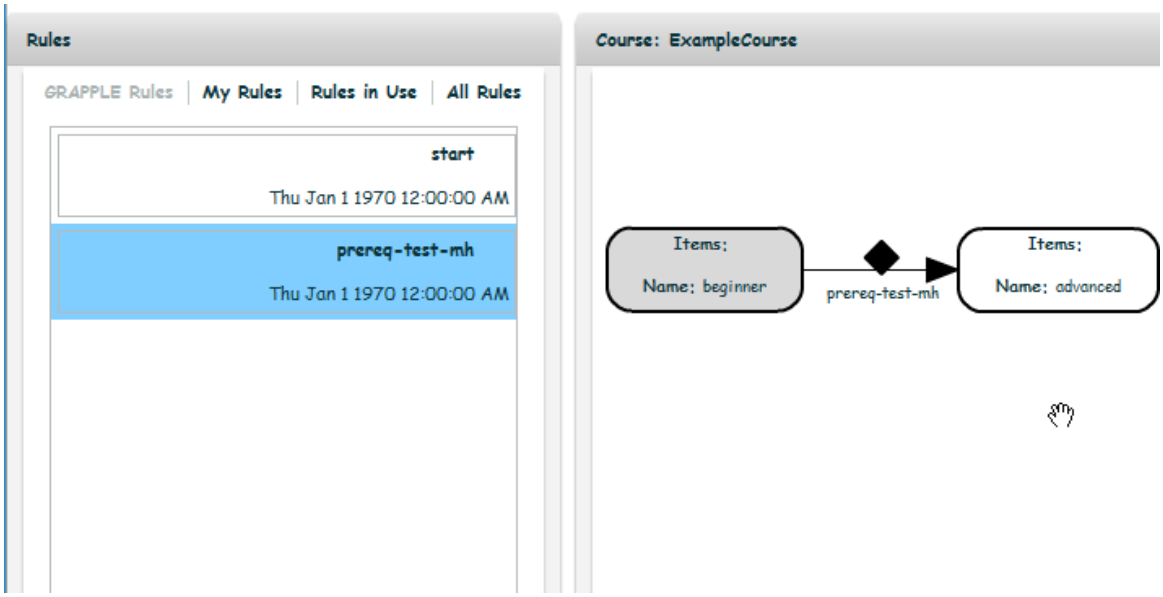

Figure 10. The prerequisite CRT, which has 2 sockets, inserted into the CAM

In addition to the previous method (Figure 11), CRTs can also be copied and pasted. Copying a CRT is done by right-clicking on a CRT node and selecting "*Copy CRT*". Next the CRT can be pasted by right-clicking on an empty space on the canvas and choosing "*Paste CRT*". This pastes the CRT instance, including modified colours and shapes, but the sockets will not contain any concepts. See Figure 11 and Figure 12. Please note that CRT instances can be copied from other CAM windows as well as the current window.

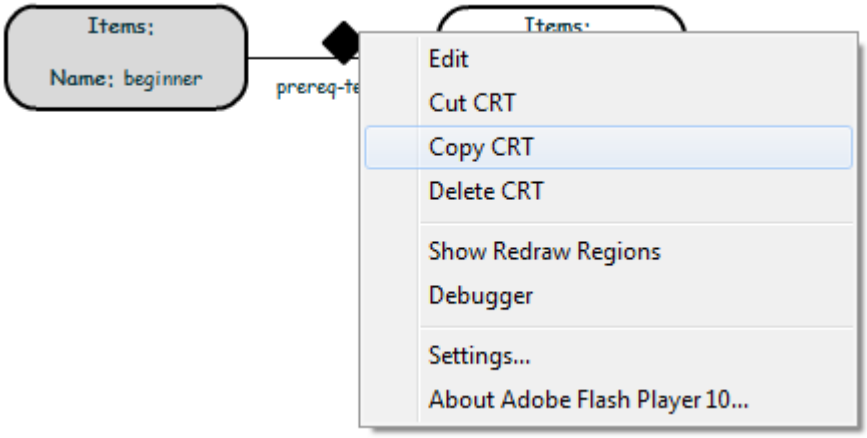

Figure 11. Copy a CRT instance. Other actions are delete and cut (copy then delete)

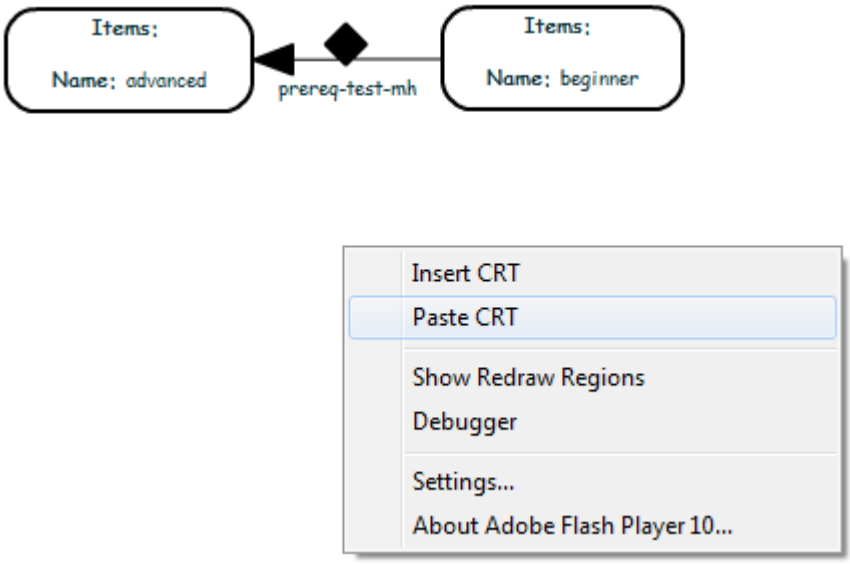

Figure 12. Paste CRT is now enabled

5.2 Filling sockets with concepts

The author now has to fill all sockets with at least one concept. There are various ways of doing this:

- Copy-paste concepts from a CAM window (note: this can be a different CAM window from the current one) see Figure 13, Figure 14 and Figure 15.
- Copy-paste concepts from a Domain Model window (see section 5.2, Figure 14 and Figure 15).
- Through the edit socket interface (see Figure 15, Figure 16, and Figure 17). This also allows insertion of external locations (see Figure 18).

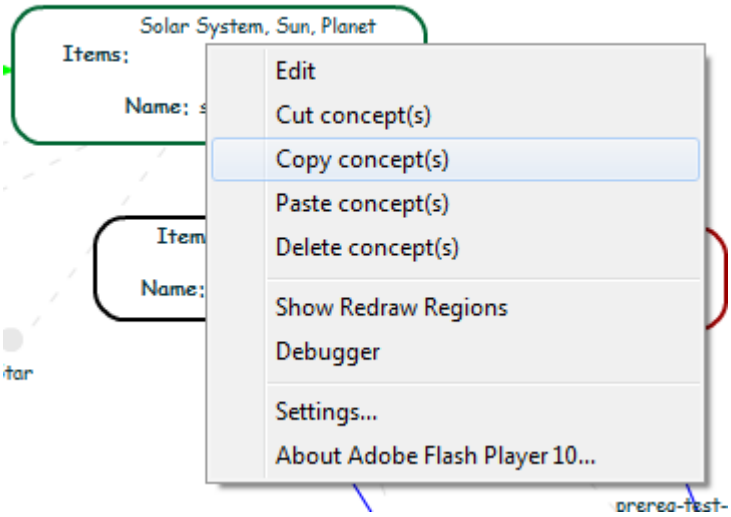

Figure 13. Copy concepts from a socket in a CAM window

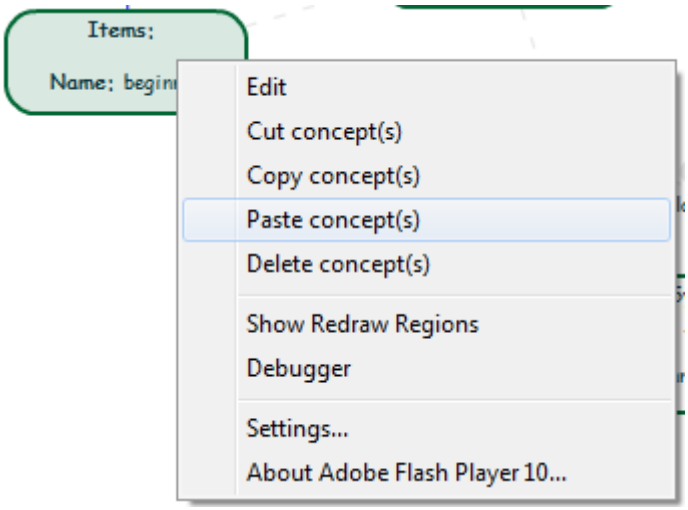

Figure 14. Pasting Concepts into sockets

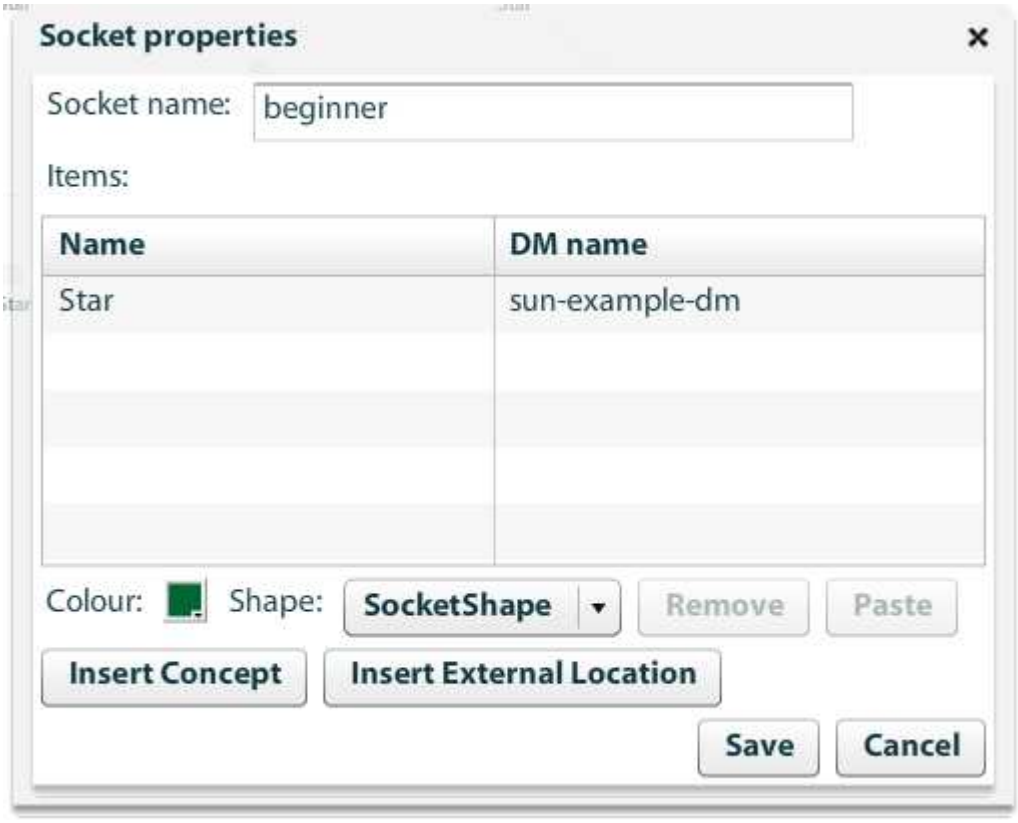

Figure 15. Socket editing screen providing an alternate space to paste concepts and an interface to insert concepts

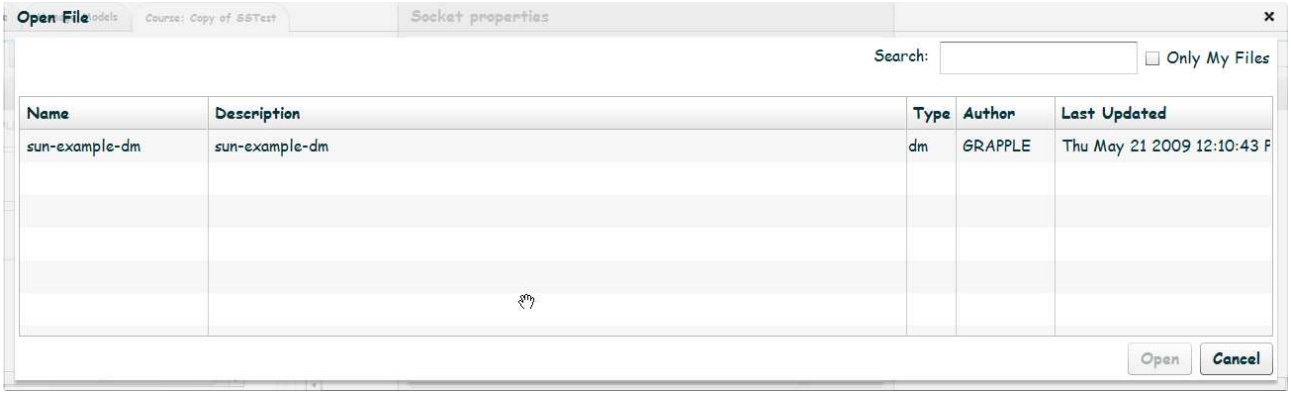

Figure 16. Selection interface for the Domain Model the concept is coming from

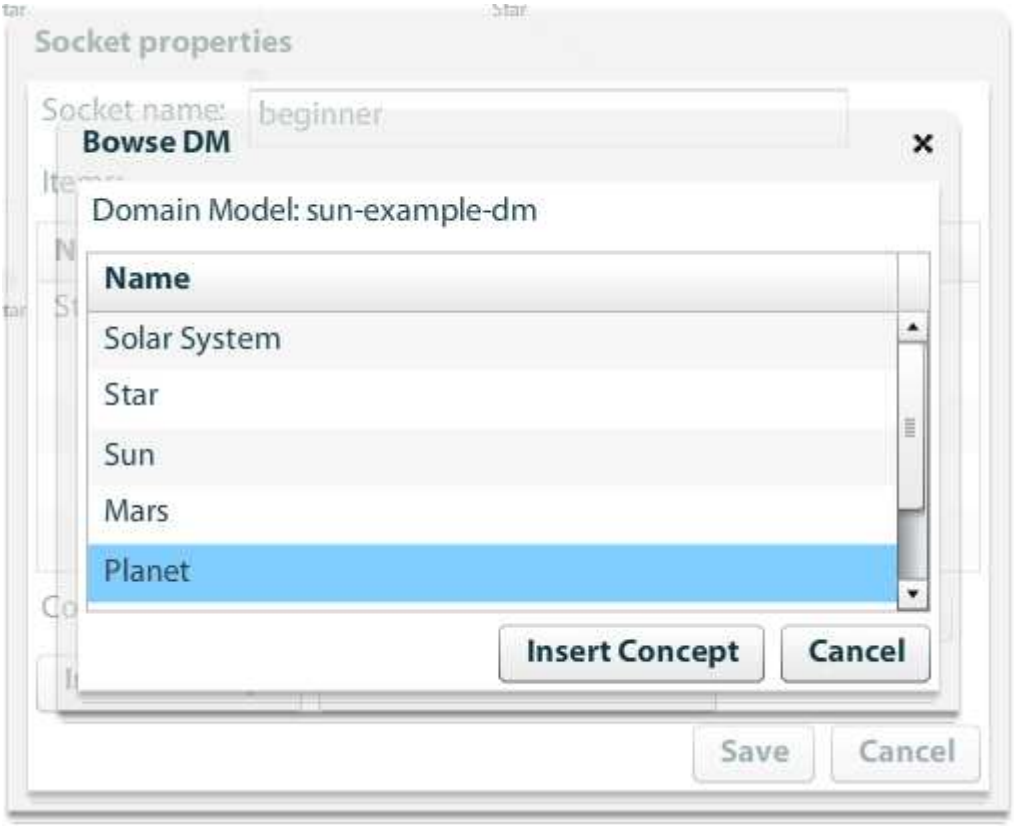

Figure 17. Selection of the Concept

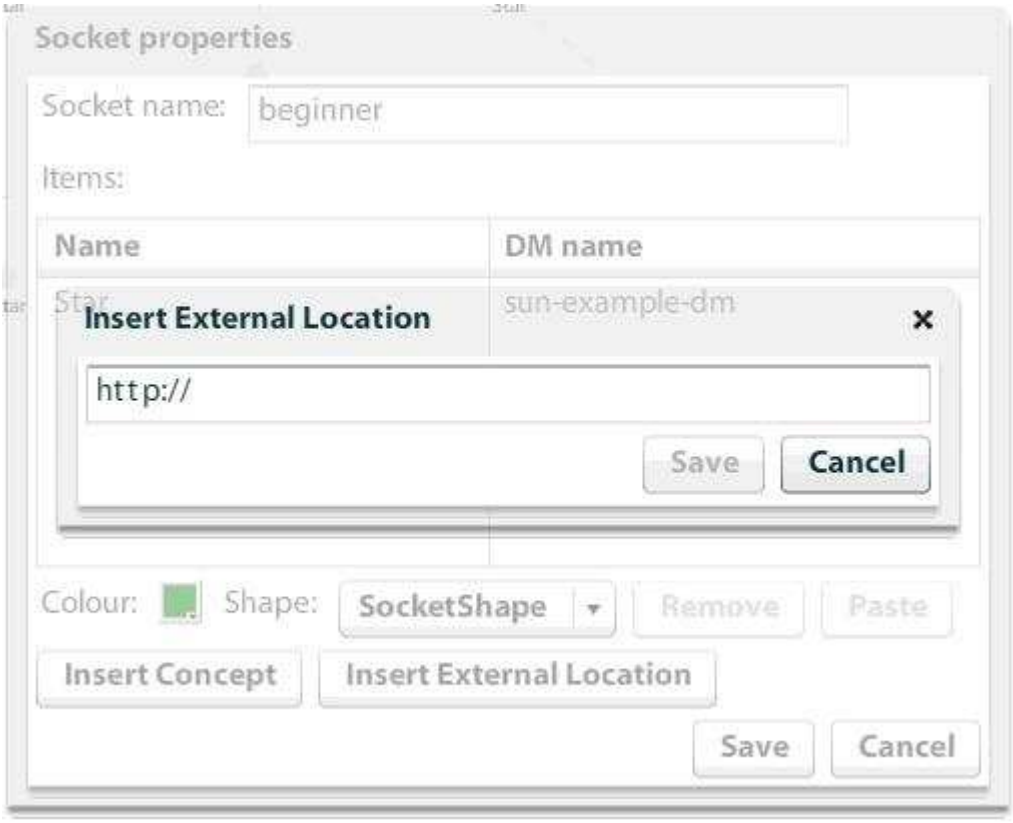

Figure 18. Insert External Location

Please note that the tool will automatically link sockets that have some concepts in common in order to indicate this relationship (see Figure 19).

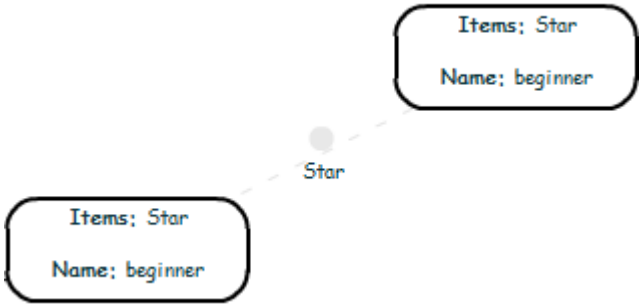

Figure 19. Two sockets containing the concept Star

5.3 Editing CRTs and Sockets

CRTs and Sockets can be edited by double clicking or by right-clicking on the socket or CRT node and choosing edit.

The screen for editing properties of sockets is shown in Figure 15. It offers the possibility to change the name, shape and colour of the socket. It also allows pasting, removing and inserting concepts and external locations.

The screen for editing CRT instances is shown in

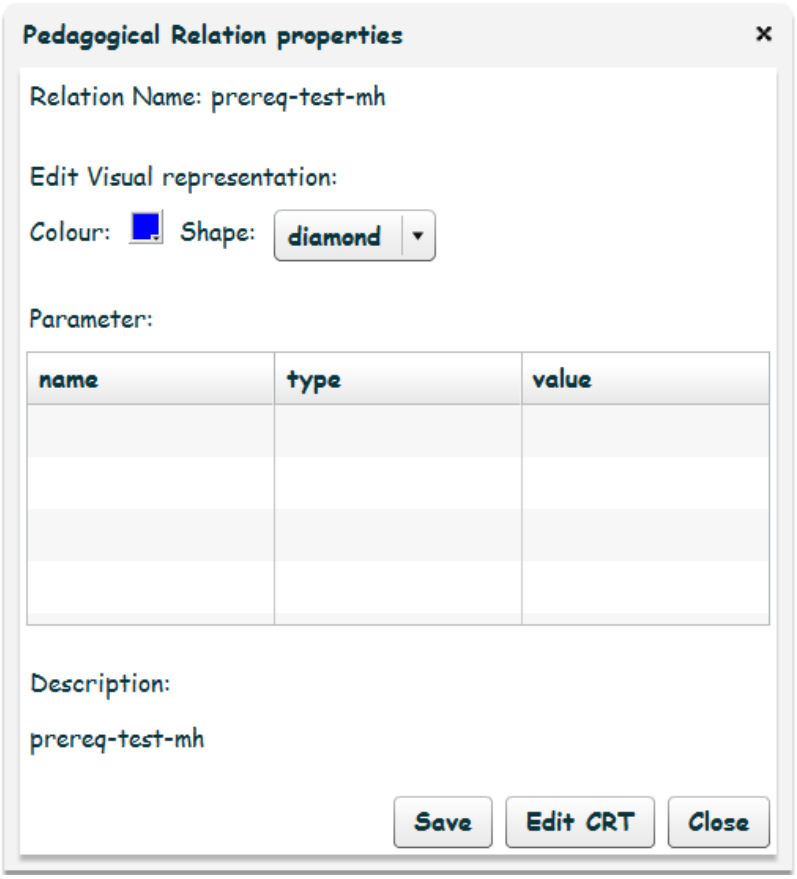

Figure 20. The dialog allows changing the colour and shape of the CRT node. It also allows launching a new window of the CRT-tool with the description of this CRT instance loaded for editing. Finally it allows updating the CRT description in this model for this type of CRT from a CRT description stored in the repository.

In order to make changes to the description of the CRT and ultimately to the adaptation code, in this CAM model an author has to open the CRT model in a CRT-tool window. This can either be done by locating it in the menu bar under "*File -> Open"* or by clicking "*Edit CRT*" in the CRT editing screen. The author then has to make the desired changes and store the CRT description – either under the same or a different name. Finally, the author has to navigate to an instance of the CRT model that needs to be changed and click

"Update CRT". At this stage the author has to locate the CRT through a dialogue which is identical to the one under "file -> open" in the menu bar. This dialogue however only lists CRTs that will fit in this place.

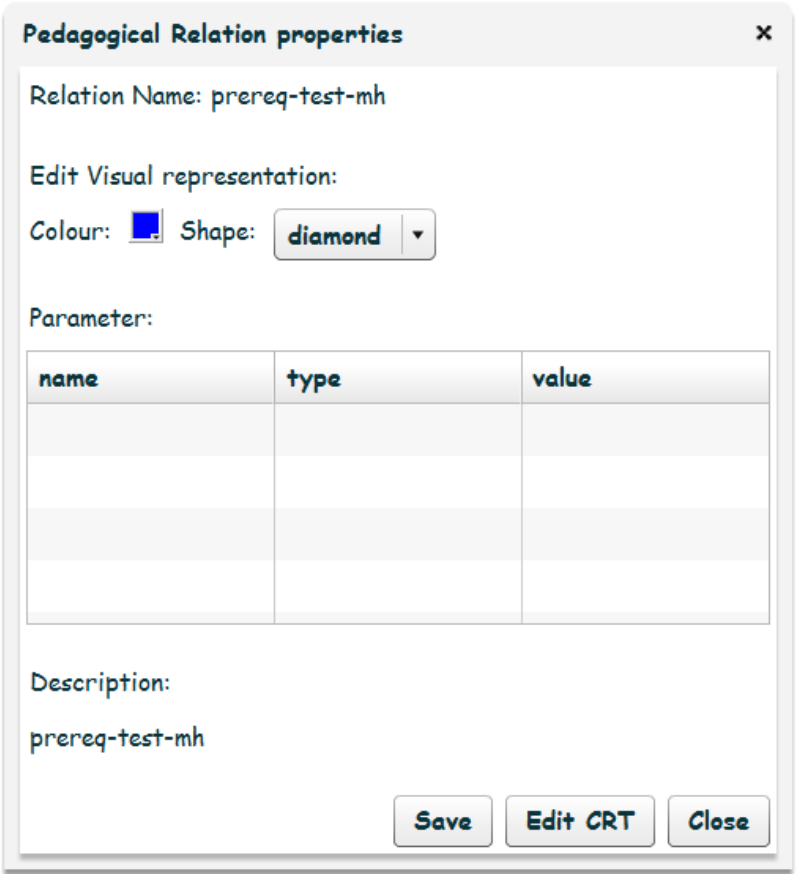

Figure 20. CRT editing screen

5.4 Deleting concepts and CRTs

Concepts can be deleted by right-clicking on a socket and selecting "*Delete concepts"* (see Figure 21). This will delete all concepts from the socket. Alternatively, in the socket edit dialogue, see Figure 15, the author can select any number of concepts and delete only the selected ones by clicking *remove* and then *save*.

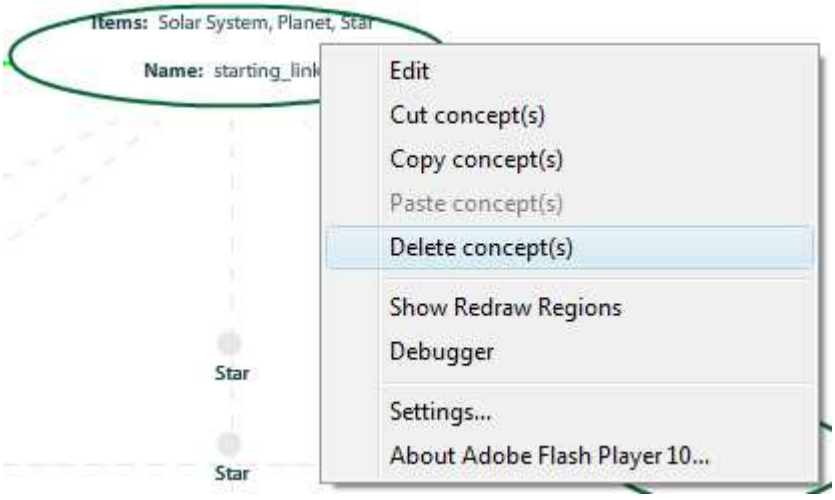

Figure 21. Deleting all concepts from a socket.

6 Conclusion and Outlook

This deliverable describes the initial development of the CAM Tool. The design has been extensively described in deliverable D3.3a. The initial version of the tool is now working and fulfils the basic requirements identified in D3.3a. CAMs can be created and modified and these CAMs can be loaded and saved in XML format to and from a database on the web. The tool is web-based and provides a graphical interface for the author.

The final version of the CAM Tool will be delivered and described in deliverable D3.2c (due 30th of July 2010). The results of the evaluation conducted in WP9 and WP10 will be taken into account and influence future work on the CAM Tool. The final version will also include a number of bug-fixes, based on bugs gathered both during informal testing by the WP3 partners as well as during the evaluation. For bug tracking the GIDTS tool (http://www.grapple-project.org/gidts-1) is used. The final version will also include at least the relationshipType element as described in section 3.5. In addition to this an improved copy-paste functionality (copy buttons available under the right mouse button in the DM) and a drag and drop functionality between DM and Cam are planned.

References

1. Brooke, J.: SUS: a “quick and dirty” usability scale. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A. and McClelland, A.L. (eds.). Usability Evaluation in Industry. London: Taylor and Francis. 1996
2. M. Cannataro and A.Pugliese, “XAHM: an XML-based Adaptive Hypermedia Model and its Implementation”, 3rd Workshop on Adaptive Hypertext and Hypermedia (AH2001) in conjunction with Twelfth ACM Conference on Hypertext and Hypermedia, Arhus, Denmark, 2001.
3. A.I. Cristea, Adaptive Course Creation for All, ITCC'04, International Conference on Information Technology, Las Vegas, US, IEEE, 2004
4. Cristea, A.I., Calvi, L. The three Layers of Adaptation Granularity, UM’03, Springer.
5. A.I. Cristea, A. de Mooij, LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators, WWW’03, The Twelfth International World Wide Web Conference, Alternate Track on Education,Budapest,Hungary. 2003
6. De Bra, P., D. Smits, D., Stash, N., The Design of AHA! , ACM Conference on Hypertext and Hypermedia, pp. 133, Odense, Denmark, 2006
7. F. Garzotto and A.I. Cristea, ADAPT: Major design dimensions for educational adaptive hypermedia. ED-MEDIA 2004 Conference, 2004
8. F. Halasz and M. Schwartz, M. (1994). The Dexter hypertext reference model: Hypermedia. Communications of the ACM, 37(2), 30-39.
9. N. Koch and W. Wirsing, Software Engineering for Adaptive Hypermedia Applications? 3rd Workshop on Adaptive Hypertext and Hypermedia, UM 2004 Conference, 2001
10. Koch N., Wirsing M. Software Engineering for Adaptive Hypermedia Applications. 8th International Conference on User Modeling, Sonthofen, Germany, 2001.
11. Stash, N., Cristea, A.I., De Bra, P., Adaptation Languages as vehicles of explicit intelligence in Adaptive Hypermedia, In International Journal on Continuing Engineering Education and Life-Long Learning, vol. 17, nr 4/5, pp. 319-336, InderScience, 2007.
12. van der Sluijs, K., Hidders, J., Leonardi, E., Houben G.J.: Generic Adaptation Language for describing Adaptive Hypermedia, Proc of International Workshop on Dynamic and Adaptive Hypertext: Generic Frameworks, Approaches and Techniques (2009)
13. H. Wu, A Reference Architecture for Adaptive Hypermedia Applications, doctoral thesis, Eindhoven University of Technology, The Netherlands, ISBN 90-386-0572-2.