GRAPPLE

D7.2b Version: 2.0

Data models and related documentation - update

Document Type	Deliverable
Editor(s):	Lucia Oneto (Gilabs)
Author(s):	Lucia Oneto (Gilabs), Eelco Herder (LUH), Daniel Krause (l3s), Andrea Lorenzon (Gilabs), Kees van der Sluijs (TUE), Alessandro Mazzetti (Gilabs)
Reviewer(s):	Olga De Troyer (VUB)
Work Package:	WP7
Due Date:	M18
Version:	2.0
Version Date:	03-02-2009
Total number of pages:	29

Abstract: This document contains the description of the data models in the GRAPPLE system and the recommendations of how to exchange them within the GRAPPLE system.

Keyword list: interoperability, data model, Learning Management System, user model, data model

Summary

This document presents an update of D7.2a for a set of harmonized data models to be exchanged within the GRAPPLE system, able to guarantee interoperability within the operational infrastructure and to harmonize the data flows between the participating systems.

The great challenge of data modelling for sharing knowledge between LMS and ALE is completely under progress and not straight forward at all, since several projects and tools on the topic are still under development. We need to do pioneer work which however promises to come up with a better solution for integration and interoperability. The task of this work package is to carefully select and collect data models for GRAPPLE.

Authors

Person	Email	Partner code
Alessandro Mazzetti	a.mazzetti@giuntilabs.com	GILABS
Andrea Lorenzon	a.lorenzon@giuntilabs.com	GILABS
Daniel Krause	krause@l3s.de	LUH
Eelco Herder	herder@L3S.de	LUH
Kees van der Sluijs	k.a.m.sluijs@tue.nl	TUE
Lucia Oneto	l.oneto@giuntilabs.com	GILABS

Table of Contents

Tables and Figures

List of Figures

List of Acronyms and Abbreviations

ADL	Advanced Distributed Learning
ALE	Adaptive Learning Environment
CAF	Common Adaptation Format
CAM	Conceptual Adaptation Tool
CDM	Concept Domain Model
CRT	Concept Relationship Type
DM	Domain Model
ELEX	Enterprise Learn eXact
GAL	GRAPPLE Adaptation Language
GALE	GRAPPLE Adaptive Learning Engine
GAT	GRAPPLE Authoring Tool
GCC	GRAPPLE Conversion Component
GRAPPLE	Generic Responsive Adaptive Personalized Learning Environment
GUMB	GRAPPLE User Model Broker
GUMF	GRAPPLE User Model Framework
IMS LIP	IMS Learner Information Package
IMS VDEX	IMS Vocabulary Definition EXchange
LMS	Learning Management System
LOM	Learning Object Metadata
RDF	Resource Description Framework
RT	Relationship Type
SCORM	Sharable Content Object Reference Model
SDM	Service Domain Model
SPARQL	SPARQL Protocol and RDF Query Language
SRT	Service Relationship Type

SVG	Scalable Vector Graphics
UM	User Model
UML	User Modelling Language
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
WP	Work Package
VR	Virtual Reality
XML	eXtensible Markup Language

1 Introduction

This document presents the update of the harmonized data models to be exchanged within the GRAPPLE system, proposed in the previous version of D7.2, able to guarantee interoperability within the operational infrastructure and to harmonize the data flows between the participating systems.

The GRAPPLE project has to guarantee the cooperation of internal components of the Adaptive Learning Environments (ALE), and Learning Management Systems (LMS). In order to reach this challenging task, suitable data models for interoperability are needed and consolidated.

The first part of the deliverable provides a short introduction of the document that is a simple update of D7.2a delivered in the first half of the year. The second part describes in detail the data models produced in the GRAPPLE framework during the second year of the project and how they can interact to each other.

1.1 Task and Deliverable Description

T 7.2 Data modelling for interoperability (DFKI, GILABS, LUH)

In order to guarantee interoperability within the operational infrastructure and to harmonize the data flows in the GRAPPLE system suitable data models need to be defined and implemented, to be used by the subsystems to exchange information through their interfaces (e.g. Partial-User-Model-Exchange-Format). The design of data models will be carried out by means of a UML-based approach. For a rapid prototyping of the models and their early evaluation, suitable UML tools will be used, which allows for the semi-automatic generation of XML-like (e.g. UserML) schemas from class diagrams. This task will strongly rely on the outcomes from WP6.

D7.2.a Data models and related documentation - first version (GILABS, M9): This document will contain a first proposal for a set of harmonized data models to be exchanged within the GRAPPLE system.

D7.2.b Data models and related documentation - update (GILABS, M18): In line with the cyclical development approach adopted in the project, D7.2.a will be updated according to the outcomes from the evaluation of the first prototype of the GRAPPLE system.

D7.2.c Data models and related documentation - final version (GILABS, M27): it will contain the final specification of the harmonized data models to be exchanged within the GRAPPLE system. The main aim of this document is to serve as an interoperability guideline and reference also for future developments.

1.2 Design decisions and Course of action

This deliverable describes the current status of work conducted in the first phase of the GRAPPLE project by using the outcomes from the first prototype and the evaluation results. This version D7.2b will provide a picture on how the different data models interrelate and their difference in terms of syntax and coverage. It is strictly based on the deliverable D7.2a - Data models and related documentation - first version and will be followed by the final update D7.2c - Data models and related documentation - final version.

The second GRAPPLE prototype will be very different from the previous one: new modules will be integrated to the first prototype and a new infrastructure will be provided to support the integration process, as documented in [1].

2 Data Models in the GRAPPLE framework and their interfaces (Syntax, names, queries)

In this section, we show how the different components that are needed in order to create a course are combined. These components are:

- *Domain Model* – 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.
- *Concept Relationship Type* – the Concept Relationship Type (CRT) component contains learning related concepts, like prerequisite, specialization etc. that can be attached to the learning concepts. Furthermore, CRT contains the properties that are used in the UM, like knows, hasVisited etc.
- *Conceptual Adaptation Model* – the Conceptual Adaptation Model (CAM) is used to combine concepts from the DM and relationships from the CRT in order to provide adaptive behaviour. Therefore, CAM specifies a rule language that allow for formalization of the adaptive behaviour.
- *User Model* – the User Model (UM) stores and processes all information of a user in a semantic manner. Hence, applications can share user information and utilize reasoning functionality in order to further process and enrich user related information.
- *GRAPPLE Conversion Component* – this module (GCC) in charge to convert a set of LMS data to the IMS LIP 1.0.1 standard [16] and then manageable by the GRAPPLE framework.

The information flow between the different components in an E-Learning application like GRAPPLE Adaptive Learning Engine (GALE) is shown in Figure 1.

DM, CRT and CAM are the key elements of the GRAPPLE Authoring Tool (GAT).

The remaining part of the section is structured as follow: we first give an application scenario that outlines the process of creating an eLearning course from scratch and how the different components are utilized in such a process. Furthermore, we will elaborate on the data models of the DM, CRT; CAM and UM in more detail.

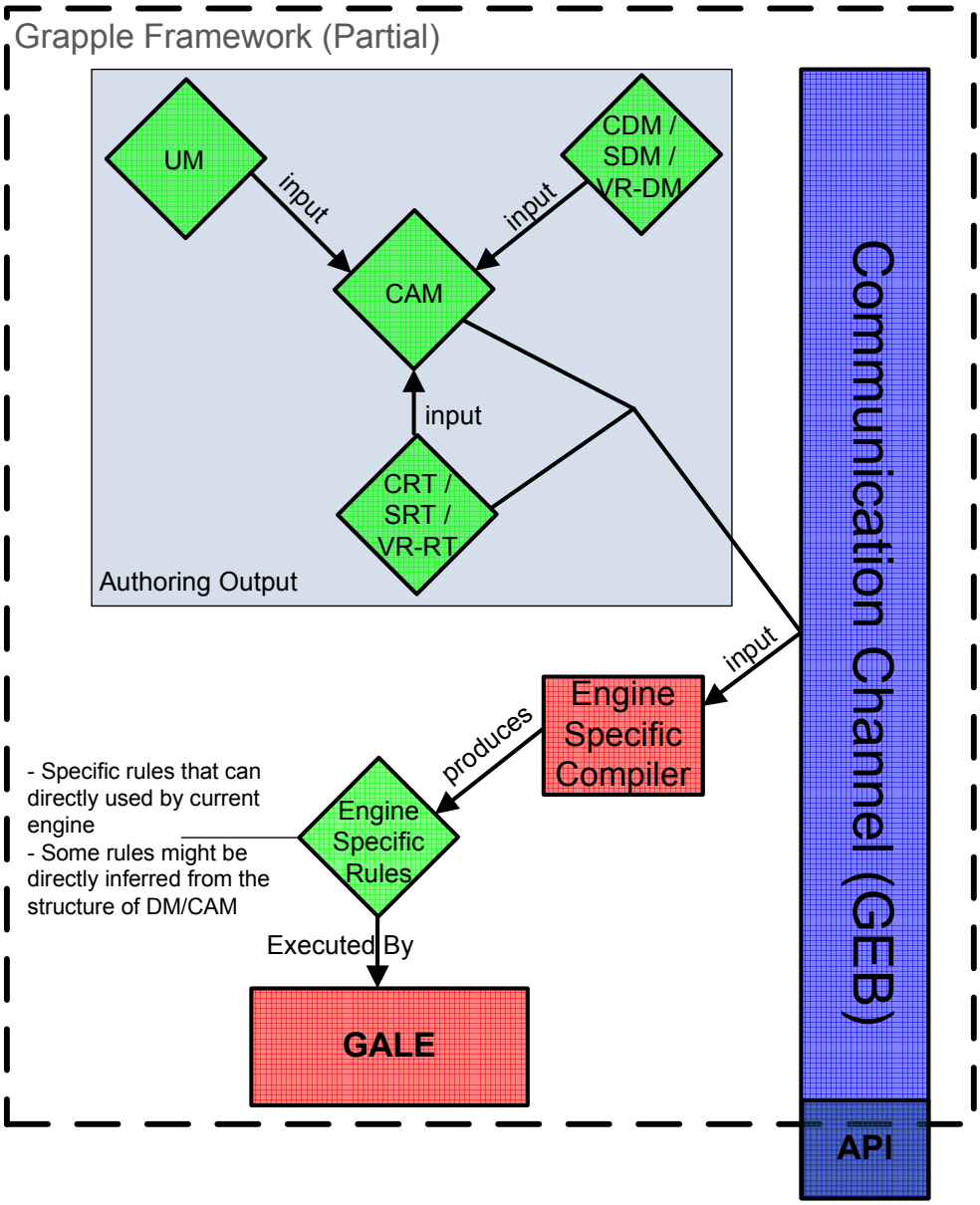

Figure 1: Information flow of GALE components and its input from the CAM (and related models).

2.1 Scenario

Pete and Lucy want to collaboratively create an eLearning course about the solar system. As Lucy is a domain expert while Pete is an E-Learning expert, they share the work appropriately:

Lucy starts to create domain concepts like earth, moon etc. and relationship between concepts, e.g. the earth is an instance of the class planets. Therefore, she utilizes the Domain Model editor in order to create concepts that are stored in the Domain Model repository. The repository uses IMS VDEX grammar, building on XML, to store the concepts.

Afterwards, Lucy hands the task over to Pete who needs to define learning related relationships that can be used with the concepts defined within the DM. These relationships are for example prerequisite, moreDifficultAs etc. Pete uses the CRT tool to create the relationships. These relationships define CRTs that are stored in the CRT library.

Afterwards, Pete utilizes the CAM tool to create an eLearning course. He accesses the CAM tool, and connects his learning resources to the concepts defined in the DM and the relationships defined in the CRT.

Afterwards, he creates some adaptation rules to express for instance that "concept X is only display if all prerequisites are known by the user".

To store which concept is known by the user, the UM is used. A CAM rule that states whenever a concept is visited by the user, the concept is marked as "known" in the UM.

Finally, the rules from CAM, the concepts from the CRT and the domain knowledge from the DM are transformed in GAL which can be used by the GALE LMS in order to provide the course.

2.2 Domain Model (DM)

The Domain Model (called also Domain Map) contains knowledge about the underlying domain that is relevant for an eLearning course. The DM tool is the main instrument to deal with Domain Models or Domain Maps.

2.2.1 Domain Map

The Domain Model tool has to pass the Domain Maps with related concepts, relationships and related attributes to the Conceptual Adaptation Model. It is important to identify which attributes describe a Domain Map:

Domain Map

- **Name**: title of the Domain Map
- **CreationDate**: creation date of the Domain Map
- **Author** of the Domain Map
- **Shared/Local**: this flag indicates if this Domain Map has been made available to all the authors in the authoring community ("Shared"). "Local" indicates the Domain Map is available only in local to his own author.
- **Description**: it provides a description of this Domain Map. Here the author can indicate the scope of this Domain Map.
- **Keywords**: they are words used in a reference work to link this Domain Map to other Domain Maps
- **Identifier**: it identifies the single Domain Map. A unique identifier is extremely important when the Domain Map is shared in the authoring community.
- **Concepts**
    - **Name**: title of the Concept
    - **Identifier**: it identifies the single concepts
    - **Description**: it provides a description of this concept
    - **Keywords**: they are words used in a reference work to link this concept to other concepts
    - **Resources**: they are resources (content) that can be used for this concept
        - **Name**: name of a single resource
        - **Location**: it indicates the path where the resource is available.
        - **UsageType**: any resource can be used in a different context such as introduction, image, body text, conclusion, ..
        - **LOM metadata**: these are the standard LOM metadata used to manage resources
- **Relationships**
    - **Name**: name of the relationship.

Any Domain Map has to be composed of at least one concept and the concept can be connected by binary relationships.

2.2.2 Vocabulary of concept and concept types

The IMS Vocabulary Definition Exchange (VDEX) specification defines a grammar for the exchange of value lists of various classes: collections often denoted "vocabulary". Specifically, IMS VDEX defines a grammar for the exchange of simple machine-readable lists of values, or terms, together with information that may aid a human being in understanding the meaning or applicability of the various terms. IMS VDEX may be used to express valid data for use in instances of IEEE LOM, IMS Metadata, IMS Learner Information Package and ADL SCORM, etc, for example. In these cases, the terms are often not human language words or phrases but more abstract tokens. IMS VDEX can also express strictly hierarchical schemes in a compact manner while allowing for more loose networks of relationship to be expressed if required.

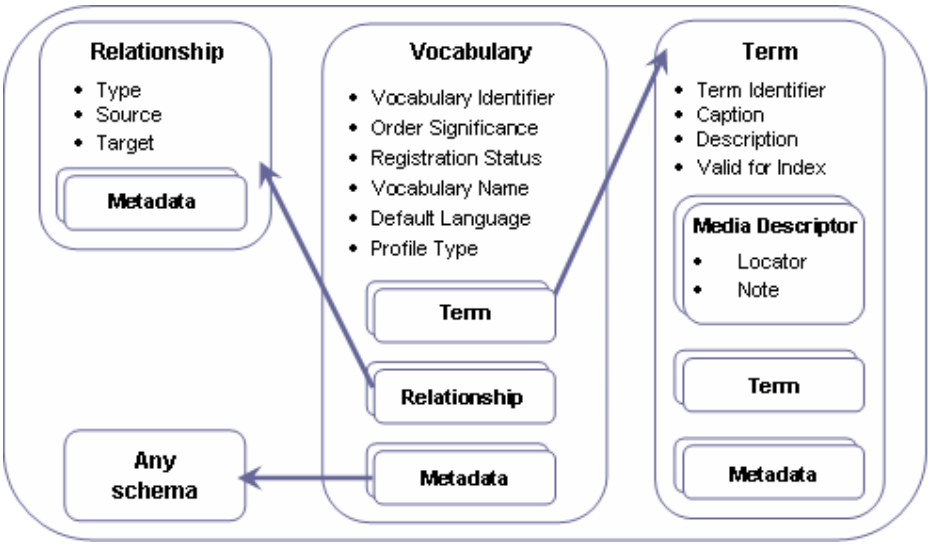

Figure 2: VDEX Information Model.

Thus the domain model provides learning objects enriched with metadata and interlinked with related concepts for an adaptive e-learning system. Based on the current requirements, specified by metadata and concepts, the system can retrieve relevant learning objects to be provided to the learner. Learning objects distributed in various repositories with associated metadata can nowadays be retrieved by means of federated search 2. Early adopters have already started using these services. These users can be either learners via their learning environments, or teachers that need suitable materials to support their classes and possibly applying blended learning approaches. From the pedagogical point of view a seamless integration of content- and concept-based navigation, based on the domain model itself, can support cognitive flexibility and foster learning.

2.2.3 GRAPPLE Domain Model and IMS VDEX

IMS VDEX standard represents the format of the Domain Model structures elaborated by the Domain Model component and stored in the Domain Model repository. This standard has got a grammar that can be used in the GRAPPLE project context.

Figure 2 identifies the key elements of the IMS VDEX grammar:

- Vocabulary: it corresponds to the GRAPPLE Domain Map
- Term: it corresponds to the GRAPPLE concept
- Relationship: it matches with the GRAPPLE relationship.

The Common Adaptation Format (CAF) encodes Domain Maps that need to encode at least one concept, which has lists of attributes and resources. A sample of an IMS VDEX file is given below:

```
<vdex>
   <term x="120.5" y="24.5">
      <termIdentifier>4FDFB0ED-3F11-960E-AC5F-002413333ABE</termIdentifier>
```

```
        <caption>
            <langstring language="it">Via Lattea</langstring>
        </caption>
        <description>
            <langstring language="en">Top level concept.</langstring>
        </description>
        <metadata>
             <resource id="BD49CBED-9D55-4545-299C-7AF210B5E4BE">
                <lom>
                    <general>
                        <title>
                            <langstring>new Title</langstring>
                        </title>
                        <description>
                            <langstring>new Description</langstring>
                        </description>
                    </general>
                </lom>
            </resource>
            <resource id="BD49CBED-9D55-4545-299C-7AF210B5E4BE">
                <lom>
                    <general>
                        <title>
                            <langstring>new Title</langstring>
                        </title>
                        <description>
                            <langstring>new Description</langstring>
                        </description>
                    </general>
                </lom>
            </resource>
        </metadata>
        <mediaDescriptor>
            <mediaLocator>http://en.wikipedia.org/wiki/Milkyway</mediaLocator>
            <interpretationNote>
                <langstring language="x-none">BD49CBED-9D55-4545-299C-7AF210B5E4BE</langstring>
            </interpretationNote>
        </mediaDescriptor>
    </term>
    <term x="165" y="147">
        <termIdentifier>1B3EF646-F051-EB59-47D1-0024B97CAC18</termIdentifier>
        <caption>
            <langstring language="it">Stella</langstring>
        </caption>
        <description>
            <langstring language="en">Star as an abstract concept.</langstring>
        </description>
        <metadata/>
    </term>
.....
    <relationship>
        <sourceTerm>1B3EF646-F051-EB59-47D1-0024B97CAC18</sourceTerm>
        <targetTerm>4FDFB0ED-3F11-960E-AC5F-002413333ABE</targetTerm>
        <relationshipType source="http://www.grapple.org/relations.xml">belongsTo</relationshipType>
        <metadata/>
    </relationship>
    <relationship>
        <sourceTerm>1C2C9337-4E9A-DD77-FD26-002506895E4A</sourceTerm>
        <targetTerm>1B3EF646-F051-EB59-47D1-0024B97CAC18</targetTerm>
        <relationshipType source="http://www.grapple.org/relations.xml">Produces</relationshipType>
        <metadata/>
    </relationship>
```

</vdex>

2.3 Concept Relationship Type (CRT)

In the Concept Relationship Type, the pedagogical relationships are described. Therefore, a language for specifying CRTs is needed. This language is composed of a vocabulary (section 2.3.1) and a format (section 2.3.2). In section 2.3.3, we specify the (web) interface to access and maintain the CRTs.

2.3.1 Vocabulary of concept relationships

In order to specify a CRT a controlled vocabulary is needed. As described in D3.2a [3], a CRT can be defined according to a data model. Since the CRT definition still can change in its details, only a preliminary vocabulary can be given in this version of this deliverable.

```
entity type = [ anchor | target ]
```

where *anchor* defines the source concept and *target* defines the concepts where the CRT is pointing to.

```
entity = [ concept | service | resource ]
```

```
cardinality = [ binary | n-ary | group | binary-group | n-ary-group ]
```

2.3.2 Format of the concept relationships

In general the CRT tool exchanges data indirectly via the CRT library located in a Web Service. It stores CRTs in this library which makes these CRTs available for other tools. A data format is needed for the exchange of these CRTs, in order that other tools can interpret CRTs correctly. The format will be XML-based and will look like following example below. However this format still can change over the GRAPPLE project's lifetime.

```
<CRT UUID="02934809238409384">
  <NAME>prerequisite</NAME>
  <DESCRIPTION>some description about this CRT</DESCRIPTION>
  <SHAPE>line</SHAPE>
  <COLOUR>blue</COLOUR>
  <ENTITY TYPE="ANCHOR">CONCEPT</ENTITY>
  <ENTITY TYPE="TARGET">RESOURCE</ENTITY>
  <CARDINALITY>N-ARY</CARDINALITY>
  <GAL>
     ... pieces of GAL code ...
  </GAL>
</CRT>
```

This example depicts a concrete CRT definition. The UUID specifies a unique identifier which is used to identify this CRT. Then name and description are specified in free text form. Next part is about the visual representation, shape and colour are defined. Both definitions are based on a fixed vocabulary such as predefined colours and shapes. Then the entity types are specified which this CRT is connecting and the cardinality specifies how many entities are connected. Entity type and cardinality are specified by using a fixed vocabulary. Last part is a piece of GAL code which defines the adaptive behaviour of a CRT.

2.3.3 Interface

Since the CRT tool mainly communicates indirectly with other tools via a Web Service, the most important interface to define is the Web Service interface. Other tools can access the CRT library via this interface.

- String[] getCRTs ()
  returns an array of all available CRTs (the identifies of the CRTs)
- String getCRT (String uuid)
  returns the XML representation of a given CRT

- void modifyCRT (String uuid, String crtxml)
  modifies a CRT specified with an identifier
- void deleteCRT (String uuid)
  deletes a CRT specified with an uuid
- String addNewCRT (String crtxml)
  adds a new CRT into the CRT library and returns the uuid for the new CRT
- String[] getDMRelationships (String uuid)
  returns all semantic relationships which are used in the GAL code by the given CRT specified with uuid
- String[] searchCRT (String searchstring, String attributename)
  searchs CRTs which contain a given string in a given attribute

2.4 Conceptual Adaptation Model (CAM)

The Conceptual Adaptation Model is used by course editors to specify relationships between concepts. These relationships will drive the adaption later on. Therefore, instances are defined by utilizing the DM and are coupled by applying relationships defined in the CRT tool.

2.4.1 The CAM languages

In order to describe adaptive behaviour in the scope of the CAM model, the following languages are defined:

1. The *CAM visual language*, the drag and drop language for authors to create adaptive story lines, (i.e. CAM instances)
2. The *CAM internal language*, the XML representation of the CAM visual language, and thus used for the internal representation of a CAM instance,
3. The *CAM external language*, for export, based upon the CAM internal language, but with every concept bound.

This apparent profusion of languages will allow the authors with different proficiency levels to describe adaptation, as will be explained in the following sections. The rest of this section will sketch each of these languages in more detail.

The CRT language [3] is not meant for daily usage, but is a description language of the adaptive behaviour, which will be encapsulated and represented by a graphical symbol in the CAM tool, or by elements such as the *relationship* in the CAM languages. The CRTs are expressed in an XML-based format, containing snippets of GAL (GRAPPLE Adaptation Language) code [5]. The GAL strives to be an implementation independent Adaptation Language, and is also intended to be independent of the authoring framework used.

For the scope of the CAM tool and language, the number and type of entities that can be connected to a CRT instance is important, as well as any restriction on combinations. Moreover, the metadata describing the CRT behaviour in layman's terms is vital for the non-programmer author, to be able using CRTs efficiently in the CAM tool. Thus, the CRT language needs to be able to express, in an easily accessible way, data that is useful for the CAM tool as above.

2.4.1.1 CAM visual language

The CAM visual language consists of the visual representation of CAM instances, which constitutes the result of mainly the *drag and drop* experience of the author.

The visual representation that effectively is used by 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 concepts from a DM instance, or placeholders thereof, and the links are CRT instances. There is an additional grouping operator as well.

So to summarise we have the following components that make up the graphical representation:

- *DM Concepts*, as in the DM, or placeholders thereof, with default representation:

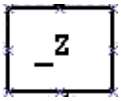

Figure 3: Placeholder for a concept.

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

Figure 4: Example of prerequisite CRT representation in CAM visual language.

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

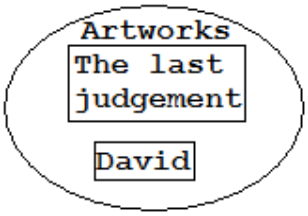

Figure 5: Grouping of concepts.

All of these defaults can be replaced by images from a library, as well as by simple descriptors, as defined by the CAM internal language [4]. For more examples of CAM instance descriptions in the visual representation we refer to [4].

2.4.1.2 CAM internal language

The CAM internal language consists of the following parts:

A representation of the purely visual aspects; it would be highly confusing if a model when he opened later, looked different from when it was created. Therefore, there is a need to capture the representation of a model in a format. The format is XML-based and is still being extended.

A format that represents the CAM instance, hence all (links to) DMs and CRTs used and how they are instantiated. This representation still allows for placeholder concepts, with certain conditions defining which concepts are meant. As we will see in the next section, when the CAM instance is exported, all such placeholders will be replaced by their concrete matching concepts.

XML representation of visual representation

In order not to confuse the author, in all three tools, the visualisation and layout should be consistent between sessions and the DM, CRT and CAM tools. This means that a mechanism is needed to store and retrieve the graphical representation of the model, presented earlier. The ultimate decision will be made at a later stage, but something along the lines of SVG [6], that is able to deal with shapes and positions will be needed. The description in this language will be read from DM and CRT tools, in order to represent concepts or placeholders thereof, and CRTs, at the import stage.

Appendix 3 shows the current proposal for the CAM internal language, as well as an example of use. Summarizing, the schema in the appendix allows for defining the following visual information:

- For CRT instance representation: shape, colour and relative position in the CAM instance, or an image representation from the library.

- For socket (placeholder for concepts) representation: shape, colour, size and relative position to the CRT, or an image representation from the library.
- For entity (concept) representation: shape, colour, size and relative position in the socket, or an image representation from the library.
- Moreover, all of the above also allow for a default, which is given by the CRT instance specifications.

2.4.1.3 CAM external language

The CAM external language is the main format for the output of the CAM tool. It is designed to be used by other systems than the GRAPPLE authoring tool, whilst they interface with the CAM tool. The main use is by the engine-independent compiler shown in Figure 1.

Prior experience [7] 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. Thus, for the latter authors only, the CAM XML language could be potentially used directly to describe adaptive behaviour. Also prior research 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 [8], [7].

All the graphical display information of the internal language is not relevant for the export, and thus not available in the external language. Moreover, the language only needs to assign CRT placeholders of CRTs to domain model concepts, or to labels of domain model concepts, or to instances of resources.

[4] shows the current proposal for the CAM external language, as well as an example of use.

2.4.2 Definition and design of the GRAPPLE Authoring Shell

The GRAPPLE Authoring tools, the CAM tool, the CRT tool and the DM tool need an overall umbrella under which to function. Moreover, a similar look and feel is required between these tools, and the transition between tools needs to be as transparent as possible.

The GRAPPLE Authoring tool is a tool in which authors will be able to, ultimately, specify CAM instances. The author should be able to define all layers of their specific CAM instance with this tool. However, these CAM instances depend on CRT instances and DM concepts.

Thus, we have decided to split the tool into three components corresponding to the mandatory layers; DM editing; CRT editing and CAM instance editing (adaptive behaviour editing). A shell is defined to integrate these components into a single overall tool.

The user interface will look like the one described in [9] and will be described in more detail in [10].

This section describes the common infrastructure for this integration. The UML diagram is depicted in Figure 6. The shell provides three main parts:

- The *Shell* class that implements the actual shell;
- The *AbstractTool* class, an abstract class for which tools need to define a subclass, in order to be able to work with the shell;
- The configuration file, *config.xml,* where configuration settings for the tools are stored, such as the content -, CRT - and CAM instance repositories.

It was decided that communication between the components of the GRAPPLE Authoring Tool will be based on services. A number of such services will be defined (e.g., a service for the generation of a unique identifier) The description of these services will be provided in a different document to the implementers of workpackage WP3 and will guide them before the GAT prototype is available. The final form is included in [10].

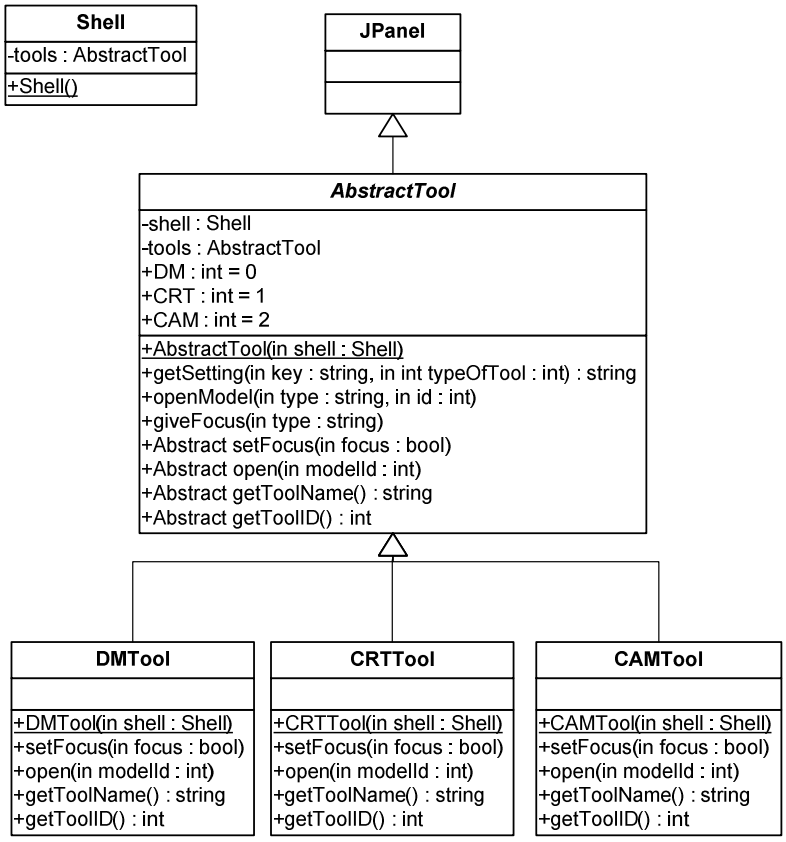

Figure 6. Shell UML diagram.

2.5 User Model (UM)

In this section we will present the core user profile format that will be used in GRAPPLE to make statements about the user/learner. These statements are called *Grapple Statements* and they may represent as many different things as preferences indicated by the user, registration of user interaction events within a system and user characteristics, as derived via one or more reasoning steps. First, we shortly repeat the approach of the user modelling framework, which deals as a broker for the Grapple statements. Afterwards, we introduce the idea that relies on Grapple statements before we present the *core Grapple User Modelling Ontology* that specifies the Grapple Statement as well as the core terms that can be used within a Grapple statement. The section ends with practical examples and an outlook regarding the role of Grapple statements within the user modelling framework.

2.5.1 User Profile Brokerage in the User Modelling Framework

From the related work we know that incorporating user profile information from other contexts is not a straightforward process. The poor take-up of the generic user modelling servers, developed in the 1990s, suggests that a centralized approach, with predefined ontologies, does not cater the needs of the multitude of adaptive systems, which are very heterogeneous in nature. Incorporating external user data should be done with as much (or even more) care as building a model from own data. In addition to uncertainty on user observation, inference and representation of the data, uncertainty due to change of context should be taken into account [11].

Based on the above, we designed a framework that facilitates the *brokerage* of user profile information and user model representations. This framework, which we call the GRAPPLE User Modelling Framework (GUMF), is designed to meet the following requirements. First, various types of systems should be able to connect to the framework. Further, the framework should provide a *flexible user model format* that allows for new types of statements and derivation rules. Sufficient *metadata* should be given to indicate its origin, contents and validity. The browsing and searching of user data or model extensions, provided by the connected systems, should be supported by *rating mechanisms*. Several systems may provide *competing* models of, for example, user interests, and as the quality of these models can vary significantly. It is important that a system administrator (i.e. a user of the framework) can take a motivated decision which alternative is most suitable for his personalization purposes.

The core element of the framework can be considered a *broker*, which provides the means for other systems to share and make use of their user data. In this section we provide an overview of the elements that are needed for setting up this framework.

In Figure 7 a generic overview of the GUMF architecture is depicted. The central element of the framework is the GRAPPLE User Modelling Broker (GUMB), which manages the communication between the connected systems. The broker keeps track of the registered systems, the available user model data and ontology extensions. Further, it keeps a centralized repository of user events. The framework provides Web-based administrative interfaces for *managing* the system configuration and for *exploring* the available user data streams, reasoning mechanisms and ontology extensions. The target audience of these interfaces consists of the administrators and programmers of client (adaptive) systems, in order to find and incorporate suitable user data streams and to offer their own data streams. Once configured, the client systems can exchange user data without human intervention.

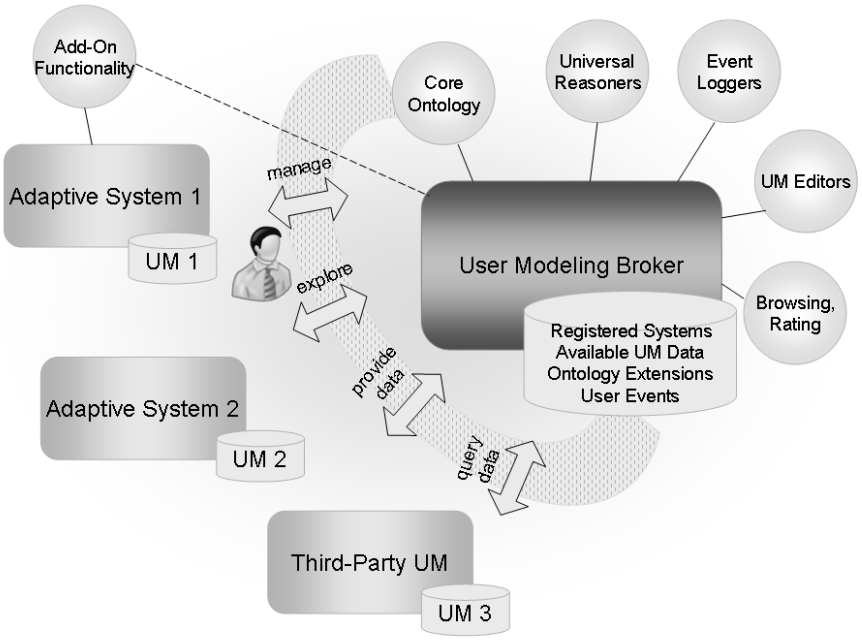

Figure 7 - User Modelling Broker Approach.

The main objective of the GUMF is to derive/infer new Grapple statements from the existing ones. The GUMF receives Grapple statements that, for example, describe user profiles and user observations from the registered client applications. The term of registered client application refers to software that is registered and communicates with the GUMF. The applications are allowed to define their derivation rules that are used by the GUMF to derive new Grapple statements from the existing ones that are maintained by the GUMF. Also, the applications are able to provide schemas that summarize types and properties of the statements. The new derived Grapple statements can be accessed by the applications in the following different ways: sending queries, subscribing to streams, or browsing.

Hence, Grapple statements build the core basis for the GUMF as they not only guarantee the exchange of user profile information (between different applications), but also enable main functionalities of the GUMF (such as deduction of new user profile information). In the next section we introduce the idea behind Grapple statements in more detail.

2.5.2 Grapple Statements

In the GRAPPLE User Modelling Framework (GUMF) a user profile consists of a set of statements, which are called *Grapple Statements* (or *Grapple Statements* or just *statements*). In a broad sense, Grapple statements can be considered as statements humans formulate in their everyday life. Thus, examples of such statements in pure textual form are:

- S1 := Mary likes chemistry
- S4 := Mary prefers learning style A to learning style B
- S6 := Mary does not like mathematics
- S7 := Mary is colour-blind
- S8 := Peter likes mathematics
- S9 := Peter is a good Java programmer (claimed by Mary)
- S10 := Peter knows the concept of object oriented programming (learned with the AHA!)
- S12 := Peter knows the Theorem of Pythagoras (learned with Moodle, last year)
- S13 := Mary does not like chemistry very much (claimed by Peter, yesterday)

In our example, one possible user model for Mary could be: M1 (Mary) := { S1, S6, S7 }. And a possible user model for Peter could be M2 (Peter) := { S8, S9, S10 }.

In GRAPPLE we model the statements above, which people usually formulate in natural language, by the notion of Grapple statements, which basically consist of a subject (usually the user), a predicate (some property of the subject), and an object (the value of the predicate). Each Grapple statement has a globally unique ID as well as some other additional properties that further describe the statement itself as well as the circumstances, in which the statement was made. Examples of these additional properties are the creation time, the creator of the statement, etc.

Grapple statements are the atomic unit of information about a user. The *user model* of a particular user is a set of selected Grapple statements. Further, Grapple statements can also state something about an entity that is not a user, e.g. an organization or a group of users. Hence, in more general, a Grapple statement is the atomic unit of information about an entity of interest for user modelling.

The interesting and challenging issue of this definition of a user model is somehow hidden in the term "selected". Not all statements about a user need to be part of a user model. That is especially necessary for contradicting statements. A decision has to be made at the time when the user model is constructed. Which "selected" statements belong to a user model is decided in a possibly complex algorithm by the user modelling framework. The focus in this section is "how can such information be syntactically defined and represented for ALE and LMS applications"?

This first step is decomposition: the accretion method shows the way to decouple user profiles or user models into smaller parts, into information units, which are formed by the Grapple statements. Since a user profile is a set of Grapple statements, the user profile format can be led back to the Grapple statement format, which bases on a set of terms that form the Grapple Core User Modelling Ontology that is described in D2.1 [12].

2.5.3 Usage of Grapple statements in the User Modelling Framework

As mentioned above, Grapple statements are the user information units the GRAPPLE User Modelling Framework (GUMF) deals with. In the following two subsections we illustrate how the GUMF utilizes Grapple statements.

2.5.3.1 Adding Statements, Derivation Rules and Schemas

The registered client applications are able to add statements and derivation rules to the GUMF. Then, internally, the GUMF stores these statements and derivation rules. Note that the GUMF also allows the applications to modify and remove the statements and derivation rules. The set of statements is logically divided into subsets that we call Grapple dataspaces or simply dataspaces. These are subsets of the set of statements which contain exactly all statements of some creator, i.e., were added by the same client application.

With each of these dataspaces also a set of derivation rules is stored which is specified by the client application in question. These derivation rules are called Grapple derivation rules (or simply derivation rule in this section) and define how additional statements are derived in this dataspace from other statements in this or any of the other dataspaces. These rules will be able to express simple types of inference in terms of premise-conclusion rules that derive new statements from the existence of other statements. These other statements can be in other dataspaces and may include derived statements. The rules will allow the derivation of aggregating statements and the derivation of common statements from registered events. The derived statements will be added only to the dataspace with which the rule is associated. This gives each client application full control over its own dataspace. It can, for example, declare by specifying certain rules that statements from certain other dataspaces are partially or fully included in its own dataspace. The derivation rule language will be extensible in the sense that special extra predicates over statements and components of statements can be added which can then be used in the premises of the rules. These predicates might capture for example special algorithms for deciding whether two learners in two different dataspaces are in fact the same person.

Beside the statements and derivation rules, the GUMF also allows the applications to add, remove, and modify schemas or parts of schemas that summarize, for example, which types of statements are found in a certain dataspace, and which properties of users are stored in a certain user model, or which properties of concepts are stored in a certain domain model. These schemas are stored by the GUMF and can be accessed by other client applications in order to investigate what information is available in other dataspace than their own. These schemas are either generated automatically from the statements in the dataspace, or are specified by the client application of the dataspace. In the latter case these schemas can also contain informal descriptions that describe the intended meaning of the statements.

Derivation rules enable the GUMF to generate new Grapple statements. These rules can, for example, (i) infer statements that embody new knowledge, (ii) they can be used to map between different ontologies or (iii) they describe how to solve problems where statements or rules conflict with each other. A simple derivation rule that infers new knowledge about a user might express the following: *If a user has bookmarked a website that has topic t then the user is interested in t*. Such a rule can, for example, simply be formulated as a SPARQL query:

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX gc:    <http://www.grapple-project.org/grapple-core/>
PREFIX gnop:   <http://www.grapple-project.org/nop/>

CONSTRUCT { gc:derivedStatement  gc:user      ?user .
            gc:derivedStatement  gc:predicate foaf:interest .
            gc:derivedStatement  gc:object    ?topic }
WHERE {
            ?originalStatement   gc:user      ?user
            ?originalStatement   gc:predicate gnop:hasBookmarked .
            ?originalStatement   gc:object    ?document .
            ?document            foaf:topic   ?topic . }
```

A mapping rule could simply map one value to another value or it can compose a new value from other values or decompose one value in different separate values. Conflict resolution rules can be used to define preferences among different types of statements or preferences among different rules. A more detailed description of derivation rules will be given in D2.3 [13], which presents the reasoning mechanisms within the User Modelling Framework.

2.5.3.2 Retrieving Statements

There are basically three ways that client applications can retrieve information from the GUMF. The first is a simple query interface that allows the client application to retrieve all statements from its own dataspace that match a certain simple pattern. The output is in this case always a set of statements, usually just one. This interface will be typically used to retrieve for example user model and domain model statements.

The second way to retrieve statements is by subscribing to a stream of statements, which is defined by a pattern similar to that in the query interface. As soon as the result of the query changes, because statements are added or removed, the subscribing application is sent a message for each message that is added to or removed from the query result. A typical use case for this interface is where one client application wants to be informed about user events for one of its learners, which are generated by another client application.

The third and final way to retrieve information is a browsing interface that allows the client applications to browse through the information in the different dataspaces, especially the schemas that summarize the contents of the dataspaces. The goal of this interface is to allow the course designers that work on one client application to investigate potentially interesting data in dataspaces of other client applications that might be reused and integrated into their own.

2.6 Grapple Adaptation Language (GAL)

The Grapple Adaptation Language is used to specify the adaptive navigation behaviour of adaptive applications in an engine independent way. In GRAPPLE the GAL specification is a middle step between the authoring specification and the GALE application. It specifies the whole adaptive navigation structure, i.e. the whole application except for features that are engine specific like presentation.

In section 2.6.1 the set of GAL constructs is specified that can be used to define the structure of the adaptive application (formal grammar in section 2.6.2). Within the GAL language we assume an according domain model and user model that can be queried. We expect both the domain model and user model to be accessible as an RDF dataset that can be queried via a SPARQL endpoint, where we assume that we can refer to both the DM and UM element within a query. RDF was chosen because of its flexibility and versatility as an intermediate language format. Within GRAPPLE the DM and UM format differs from both the DM and UM format in the GAT tools and the Hibernate format of the GALE engine, but given that we want to guarantee interoperability between more systems than the GRAPPLE ones, it seemed wise to choose the most generic language we could find.

2.6.1 GAL language constructs

Here we give a brief overview of the GAL constructs. For a more comprehensive overview please refer to D1.1a [5].

Units

Units can be seen as a grouping element to group those elements that semantically are grouped together on a Web page. If we consider a concrete page, then the entire page can be called a Unit as well. We refer to these units as top-level units (even though they are not discernable from other units). Within a top-level unit units are structured hierarchally via the subunit declaration.

Attribute

Attributes express the printables, i.e. the items that will be visible on screen. Attributes can print a constant text, or URL media item (e.g. picture, video, etc), but can also refer to the domain model to dynamically query a resource there.

Links

Links connects an attribute in a unit with another (target) unit. This connection means that a selection of the link by the user means that the user will navigate from the current unit to the target unit. Links allow the conveyance of information between units, e.g. to help the target unit to dynamically pick the appropriate attributes to show to the user.

Queries

Queries provide personalization and adaptation. The underlying database should provide access to both the domain model (for actual data) and the user model (for personalization) for this purpose, i.e. both accessible from the same query. Queries are not only used to fill in attribute values, but can be used to fill in every kind of value, e.g. also the name or label of an attribute, unit or link.

SubUnits

The subunit declaration allows to 'include' units in other units. Instead of fully specifying a unit in another unit, the subunit specification only points to another unit. In this way units can be reused as being subunits of various other units, as well as being toplevel units themselves. The SubUnit construction also allows passing information between the parent unit and its subunit via variables. Special types of SubUnits constructs are the SetUnit and ListUnit.

Variables

Variables can be used to convey information between units, either via the Links or the SubUnits construct. This information can be used for deciding which information to show. Where units can be designed to show information about abstract concepts, the concrete instantiation of it is typically decided on basis of variables.

SetUnits

A SetUnit is a special SubUnit construction. Instead of one reference to a Unit, a SetUnit contains a set of references to Units (typically the same unit). Passing variables then typically decided for every Unit instantiation which information needs to be shown in that Unit. A ListUnit is an ordered SetUnit, where the order of the elements in the set is of importance and preserved during presentation.

Conditions

As an additional adaptation and personalization construct, besides queries, there are conditions. A condition is basically an if-then-else constructs which allows choosing between several alternatives based on the result of a particular query. This can be used for deciding the content of an attribute, but also for instance if a subunit should be shown or not.

Updates

Adaptation is based on user behaviour, which can be recorder via update queries. An update allows to update the user model and is typically triggered by a user action. Normally updates are connected to events, but they can also be connected with links.

Events

Events are triggers of actions based on the user action. This can be used to create a Web 2.0 environment with events and scripts, but can also be used to trigger for instance updates. Examples of events are onAcces, OnExit, OnClick, etc.

Emphasis

The Emphasis construct can be used to emphasize or de-emphasize elements in the unit, or an entire subunit. It is upon engines implementing GAL how emphasis influences the presentation.

Order

The order constructs allows defining a partial order between elements within a particular unit by assigning the elements an ordering number. This order expresses for which elements in the presentation it is important the user sees them before some other elements. Any arbitrary elements within a unit can have an order attribute, but it not obliged to have one. The order attribute do not propagate through subunit, i.e. the elements within a subunit a grouped for an order attribute. Elements that have no order attribute can come in any arbitrary order in the presentation. Elements with the same order value also appear in arbitrary order with respect to each other.

Forms

Forms allow more complex information back from the user than link clicks. The user can submit either text or URLs that can be stored in the user model.

Frames

Frames allow partial navigation of a page, and also allow to define that a link click allow a unit to navigate other than the unit where the link resides in. For example, a unit can have a subunit that contains a navigation menu. Clicking one of the items in the menu could make only the main part of the main unit navigate, but not the navigation menu itself.

Scripts

Scripts allow adding specific constructs that will be treated as is by the underlying GAL engine. This allows for instance attaching scripts to the application that will only be interpreted by the user's Web browser.

Web Services

The Web Service construct in GAL allows defining a Webservice call by the engine that runs the GAL language, before any other construct in the GAL definition is evaluated. If this Web service call results in returning printables (i.e. attributes), these can be printed as part of the application.

Note that Scripts and Web Services are advanced constructs that might not be supported by all adaptive engines that might want to implement GAL. For these engines these constructs will just be ignored.

2.6.2 GAL Grammar

In this syntax we give a more formal definition of the GAL Grammar and Semantics

We assume in the following syntax that the following non-terminals already are defined: <unit-type> describes *UT*, <var-name> describes *X*, <domain-concept> describes *DC*, <literal> describes RDF literals, <val-query> describes *VQ*, <list-query> describes *LQ*, <update-query> describes *UQ*, <unit-label> describes *UL*, <unit-name> describes *UN* and <event-type> describes *ET*.

The set of GAL expressions is then defined by the following syntax:

<gal-expr> ::= <unit-decl>* .
<unit-decl> ::= <unit-type> **"["** <unit-def> **"]"** .

<unit-def> ::= **"a"** (**"gal:unit"** | **"gal:orderedUnit"**) **";"**
(<input-var>)* (<attr-def> | <subunit-expr> | <set-unit-expr> |
<list-unit-expr>)* <link-expr>? (<on-event-expr>)* **.**

<input-var> ::= **"gal:hasInputVariable" "[" "gal:varName"** <var-name> **";"**
"gal:varType" <domain-concept> **"]" ";"** .

<on-event-expr> ::= **"gal:onEvent" "[" "gal:eventType"** (**"gal:onAccess"** | **"gal:onExit") ";"**
"gal:update" <update-query> **"]" ";"**.

<attr-def> ::= **"gal:hasAttribute" "["** <name-spec>? <label-spec>? **"gal:value"** <val-expr> **"]"**.

<name-spec> ::= **"gal:name"** <elem-name> **";" .**

<label-spec> ::= **"gal:label"** <val-expr> **";" .**

<subunit-expr> ::= **"gal:hasSubUnit" "["**<name-spec>? <label-spec>? <unit-constr> **"]" ";"** .
<list-unit-expr> ::= **"gal:hasListUnit" "["** <label-spec>? <unit-constr> **"]" ";"** .

<set-unit-expr> ::= **"gal:hasSetUnit" "["** <label-spec>? <unit-constr> **"]" ";"** .
<link-expr> ::= **"gal:hasLink" "["** <link-constr> <target-spec>? **"]" ";"** .

<link-constr> ::= **"gal:refersTo"** <unit-type> **";"** <bind-list> .
<target-spec> := **"gal:targetUnit"** (<unit-name> | **"gal:_top"** | **"gal:_self"** | **"gal:_parent"**) **";"** .

<bind-list> ::= <query-bind>? <var-bind>* .
<query-bind> ::= **"gal:hasQuery" "["** <list-query> **"]" ";"**
<var-bind> ::= **"gal:assignVariable" "[" "gal:varName"** <var-name> **";"**
"gal:value" <val-expr> **"]" ";"** .

<unit-constr> ::= **"gal:refersTo" "["** <unit-def> **"]" ";"** <bind-list> .

We let each non-terminal also denote the set of expressions associated with it by the syntax, so <gal-expr> is the set of all GAL expressions.

We extend the syntax with some syntactic sugar: inside a <unit-constr> the fragment "**[**" <unit-def> "**]**" can be replaced by a <unit-type> if in the total expression this is associated with this <unit-def>. In other words, if a <unit-type> occurs in the <unit-constr> after the **gal:refersTo** then it is interpreted as the associated "**[**" <unit-def> "**]**".

For a more elaborate explanation of this syntax, or the semantics of the GAL constructs refer to D1.1b [14].

2.7 Current Implementation Status

This section describes the current status of the implementations of the single components.

2.7.1 Domain Model (DM)

For the DM, a specification of the language, which is based on XML, exists. Specifications of the interfaces are specified with the other GAT tools (CRT and CAM). The first version of the DM prototype is available online and can be accessed via:

http://prolearn.dcs.warwick.ac.uk/GAT-mockup/.

2.7.2 Concept Relationship Type (CRT)

For the CRT, a specification of the language, which is based on XML, exists. Web services interfaces to access the data have also been specified. A first implementation is available:

http://prolearn.dcs.warwick.ac.uk/GAT-mockup/

2.7.3 Conceptual Adaptation Model (CAM)

For the CAM, both a specification and a prototype exist. The prototype is available online and can be accessed via:

http://prolearn.dcs.warwick.ac.uk/GAT-mockup/

2.7.4 User Model

For the UM, both a specification and a prototype exists. The prototype is available online and can be accessed via:

http://semweb.kbs.uni-hannover.de:8082/grapple-umf/

2.7.5 Grapple Adaptation Language (GAL)

There exists a prototype implementation that can directly interpret and execute a GAL specification, given that the DM and UM are proper RDF datasets.

This prototype still needs to be refined to conform to the latest GAL refinements.

Moreover two compilers are in the making: one compiler that compiles the output of the GAT tools to GAL, and one compiler that compiles GAL plus the DM and UM definition to GALE GDOM files.

2.8 Open Integration Issues

Due to the early project state and the nature of the work flow, each component has been developed independently. Therefore, possible integration issues arise that are named in this section. We mainly identified two issues:

- *Inconsistency of interfaces* The naming of the methods in the interfaces of the UM and the CRT are not consistent.
- *Inconsistency of underlying standards* While CRT and DM are based on XML, UM is based on RDF. Thus, concepts and relationship that are used in the CRT and DM cannot be referenced by the UM easily.

However, as there is one component in the GRAPPLE project that interacts with all other components, namely CAM and GAL, the resolution of inconsistencies will be solved by an agreement of the interfaces of the single components and CAM/GAL.

Furthermore, we think that more crucial for integration is the consistency among the ontologies. Hence, we think that even with the actual data models integration is feasible.

2.9 GRAPPLE Conversion Component (GCC) data model

In D7.2a - Data models and related documentation - first version, we defined a minimal GRAPPLE data model able to guarantee the basis for a proper interoperability between the GRAPPLE system and the LMSs involved in the project. As already analyzed in D7.2a, Sakai, Moodle, Claroline, learn eXact[1] and CLIX[2] have got considerable differences.

D5.2a [15] identifies the possible standards to be used to support the requested interoperability and IMS LIP 1.0.1 [16].

The LMSs are a useful source of information for the GRAPPLE system: they have relevant information about the user. Then GUMF is the natural destination of all the user details necessary for the course adaptation GALE is in charge of.

The following approach has been taken in WP7: first, a set of user events has been identified. They provide user information to GRAPPLE (more specifically to GUMF):

1. *Access to a course*: it identifies the moment when the user access a course
2. *Tests/quizzes*: the user has terminated a test
3. *Registration*: the user logs for the first time in the LMS integrated with GRAPPLE
4. *Student enrolment*: the student is enrolled in a new course
5. *User login*: the user logs in the LMS integrated with GRAPPLE
6. *Role change*: the user has changed his role
7. *Learning activity change*: a particular learning activity has been changed for a given user
8. *Learning activity addition*: a particular learning activity has been added for a given user
9. *Learning activity removal*: a particular learning activity has been removed for a given user.

Then a list of user parameters has been prepared for each event.

The user parameters are available to the GRAPPLE system only if they are made public (see first scenario in D7.1b [1]).

The LMS is in charge to provide the user parameters to the GRAPPLE system: for the most of the LMSs it is not possible to provide the data in real time. Then it is necessary to set up a batch system able to send periodically the user parameters related to the events mentioned above.

[1] http://www.giuntilabs.com/learn_eXact_Enterprise/index.php

[2] http://www.im-c.de/en/products/

D5.2b [17] is in charge to provide the conversion components able to translate the LMS specific parameters to GRAPPLE parameters where possible. The list of user parameters prepared for any event covers all the parameters that have to be converted by the LMS specific conversion component.

2.9.1 Access to a course

The user accesses a particular course: the user can be anyone with permission rights to access a course. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Date of access: activity - 13.6 - date (13.6.1 - typename, 13.6.2 - datetime)
- Course topic: activity - 6.12 - description (13.5.2 - long)
- Course title: activity - 6.12 - description (13.5.1 - short)
- Course date of creation: activity - 13.6 - date (13.6.1 - typename, 13.6.2 - datetime).

Claroline provides Course id, Date of access, User id and Course title. Course topic is not provided. Course date of creation can be not provided.

Moodle provides Course id, Date of access, User id, Course title and Course date of creation. Course topic is not provided.

Sakai provides Course id, Date of access, User id and Course date of creation. Course title and Course topic are not provided.

CLIX provides all the above parameters.

learn eXact provides all the above user parameters only if the course is SCORM.

2.9.2 Tests/quizzes

The user (learner) completes a test or quiz. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Topic: activity - 6.12 - description (13.5.2 - long)
- Goal: activity - 6.11.12 - evaluation description
- Score: activity - 6.11.11.2 - score
- Scale description: activity -6.11.11.1 - interpretscore
- Timestamp – start (last): activity - 13.6 - date (13.6.1 - typename, 13.6.2 - datetime)
- Timestamp – end: activity - 13.6 - date (13.6.1 - typename, 13.6.2 - datetime)
- Attempts: activity - 6.11.9 – noofattempts.

Claroline provides Course id, User id, Goal, Score, Scale description, Timestamp – start (last), Timestamp – end: and Attempts. Topic is not provided.

Moodle provides Course id, Goal, Score, Scale description, Timestamp – start, Timestamp – end: and Attempts. Topic and User id are not provided. Scale description is provided as the maximum level of the scale. Timestamp – start is not the LAST, but it is the first opening of an instance of the test. Attempts: the number of possible attempts is decided by teacher and tracked.

Sakai provides Course id, User id, Goal, Score, Scale description, Timestamp – start (last), Timestamp – end: and Attempts. Topic is not provided.

CLIX provides all the above parameters.

learn eXact provides all the above user parameters only if the course is SCORM.

2.9.3 Registration

The user (learner) logs the first time in the LMS integrated with GRAPPLE: all the personal details are passed to the GRAPPLE system (GUMF is the final receiver of this data). The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Last Name: identification - 2.4 -name (2.4.4 - partname, 2.4.4.2 text)
- First Name: identification - 2.4 -name (2.4.4 - partname, 2.4.4.2 text)
- Organisation: identification - 2.4 -name (2.4.4 - partname, 2.4.4.2 text)
- Email: identification - 2.6 contactinfo (2.6.8 - email)
- Language: accessibility - 3.3 - language
- Gender: identification - 2.7 - demographics (2.7.5 - gender)
- Date of birth: identification - 2.7 - demographics (2.7.6 - date)
- Street: identification -2.5 - address (....)
- Town: identification -2.5 - address (....)
- Postal code: identification -2.5 - address (....)
- Region: identification -2.5 - address (....)
- Country: identification -2.5 - address (....)
- IP address: identification - 2.9 extension (14.13 ext_identification)
- Role: identification - 2.9 extension (14.13 ext_identification).

Role describes a group of interest to the Learning Management environment. There are many types of groups that may be shared between systems:

- *Teacher*
- *Tutor*
- *Learner*
- *System Administrator.*

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

2.9.4 Student enrolment

The user (learner) is enrolled to a course. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Role: identification - 2.9 extension (14.13 ext_identification).

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

We use an IMS LIP extension for role: the possible values are Teacher, Tutor, Learner and System Administrator.

2.9.5 User login

The user (learner) logs in the LMS integrated with GRAPPLE. This event does not correspond to the Registration event. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- IP address: identification - 2.9 extension (14.13 ext_identification).

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

2.9.6 Role change

One of the user's roles has changed. This event does not correspond to the Registration. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Role: identification - 2.9 extension (14.13 ext_identification).

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

2.9.7 Learning activity change

A learning activity has been changed. Learning activity is part of a course (a course can be composed by one or more learning activities). The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Activity id: activity - activity - 6.13 - contentype – 13.3 – sourceid (13.3.1.1)
- Status: activity - activity - 6.13 – status – 13.8 - typename (13.4). The Status identifies this event as **change**.

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

2.9.8 Learning activity addition

A learning activity has been added. Learning activity is part of a course or the course itself. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Activity id: activity - activity -6.13 - contentype – 13.3 – sourceid (13.3.1.1)
- Status: activity - activity - 6.13 – status – 13.8 - typename (13.4)). The Status identifies this event as **addition**.
-

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

2.9.9 Learning activity removal

A learning activity has been removed. Learning activity is part of a course. The following user variables are identified and mapped to IMS LIP 1.0.1 (the numbers below indicate the single table and row in the IMS LIP specification) [16]:

- LMS type (CLIX, Claroline, Moodle, ELEX, Sakai): contentype – 13.3 – sourcedid (13.3.1.1)
- url: contentype – 13.3 – id (13.3.1.2)
- User id: security key - 11.4 - keyfields (11.4.1 - fieldlabel, 11.4.2 - fielddata)
- Course id: activity - contentype – 13.3 – sourceid (13.3.1.1)
- Activity id: activity - activity -6.13 - contentype – 13.3 – sourceid (13.3.1.1)
- Status: activity - activity - 6.13 – status – 13.8 - typename (13.4). The Status identifies this event as **removal**.

Claroline, Moodle, Sakai, CLIX and learn eXact provide all the above parameters.

3 Conclusions

This deliverable is a considerable update on the data models developed during the second year of the GRAPPLE project. Many GRAPPLE components have been introduced to the overall GRAPPLE infrastructure and harmonised data models are strongly needed: in this document it is possible to have a general overview of them. A third update is scheduled during the last year of the project and it will contain the final list of the data models of the GRAPPLE framework.

References

1. Oneto L. et all: D7.1b - Updated specification of the operational infrastructure
2. Simon, B. and all, A Simple Query Interface for Interoperable Learning Repositories *WWW 2005* , May 10-14, 2005.
3. Steiner C., Nussbaumer A.: D3.2a - Integrated model of adaptation on learning with specifications
4. Hendrix M., Cristea A.: D3.3a - Design of a CAM definition tool
5. van der Slujis K. at all: D1.1a - CAM to adaptation rule translator – specification
6. Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003, http://www.w3.org/TR/SVG11/
7. Stash, N., Cristea, A.I., and De Bra, P., Explicit Intelligence in Adaptive Hypermedia: Generic Adaptation Languages for Learning Preferences and Styles, Proceedings of the HT 2005 CIAH Workshop, Salzburg, 2005
8. Cristea, A.I., De Bra, P., Explicit Intelligence in Adaptive Hypermedia: Generic Adaptation Languages for Learning Preferences and Styles, HT 2005 CIAH Workshop, Salzburg. (2005)
9. Oneto L., et all: D3.1 - Design specification of a DM definition tool
10. Hendrix M., et all: D3.3b - Initial implementation of the CAM definition tool
11. Herder, E.: Forward, Back and Home Again - Analyzing User Navigation on the Web. Ph.D. Thesis, University of Twente. ISBN 907383873-8. (2006)
12. Herder E., Abel F.: D2.1 - Definition of an appropriate User profile format
13. Herder E.: D2.3 - Tool for reasoning about user observations
14. van der Slujis K. at all: D1.1b - CAM to adaptation rule translator – implementation
15. Oneto L., et all: D5.2a - Conversion models between GRAPPLE and LMSs
16. IMS Learner Information Packaging Information Model Specification, January 2005, **http://www.imsglobal.org/profiles/**
17. D5.2b - Conversion components between GRAPPLE and LMSs.