GRAPPLE

D7.2c Version: 1.0

Data models and related documentation - final version

Document Type	Deliverable
Editor(s):	Alessandro Mazzetti
Author(s):	Alessandro Mazzetti, Kees van der Sluijs, Michele Dicerto, Paolo Tenerini, Alexander Nussbaumer, Jonny Foss, Fabian Abel, David Smits
Reviewer(s):	Jan Hidders (TUD), Cord Hockemeyer, Alexander Nussbaumer (both UniGraz)
Work Package:	WP7
Due Date:	31-5-2010
Version:	1.0
Version Date:	22-07-10
Total number of pages:	26

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.

The described data models are relevant to: GRAPPLE Event Bus (GEB), Domain Model (DM), Concept Relationship Type (CRT), Conceptual Adaptation Model (CAM), User Model (UM), GALE code (GAL), GRAPPLE Conversion Component (GCC),

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

Summary

This document is an update of D7.2b and contains the data models definition and the documentation of the final version of GRAPPLE software.

Section 2.1 refers to the GRAPPLE Event Bus, which is the unifying framework managing the communication system among the various GRAPPLE components. Several operations are described.

Section 2.2 refers to the Domain Model component, having few modifications since the previous version, from the data modelling point of view.

Section 2.3 refers to Concept Relationship Types, dealing with several aspects: CRT dialects, Visual representations, Sockets, Behaviours, Constraints and Associations. In order to exchange CRTs with other GRAPPLE components a data format has to be defined which can be written by the CRT Tool and read by other components. Most importantly CRTs have to be read by the CAM Tool and the GALE.

Section 2.4 refers to the CAM language, distinguishing the various language types: visual, internal and external. The data structure of the Conceptual Adaptation Model is defined in the deliverable D3.3c. In this section we describe the CAM languages as they are implemented in GAT.

Section 2.5 refers to the User Modelling with particular focus on user enrolment, role changes and test/quizzes.

Section 2.6 refers to the GAL code, addressing the parameterized GAM. GAM code relies on GALE being the actual adaptation engine. It thus inherits GALE's view on domain-, adaptation- and user-model.

Authors

Person	Email	Partner code
Kees van der Sluijs	k.a.m.sluijs@tue.nl	TUE
Alessandro Mazzetti	a.mazzetti@giuntilabs.com	GILABS
Michele Dicerto	m.dicerto@giuntilabs.com	GILABS
Paolo Tenerini	p.tenerini@giuntilabs.com	GILABS
Alexander Nussbaumer	alexander.nussbaumer@uni-graz.at	UNIGRAZ
Jonny Foss	jonny.foss@gmail.com	WARWICK
Fabian Abel	abel@l3s.de	L3S
David Smits	D.Smits@tue.nl	TUE

Table of Contents

Tables and Figures

List of Figures

List of Tables

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

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 data model here presented provides interoperability within the operational infrastructure and harmonizes the data flows among the participating systems.

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, so that they can 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.

2 Final version of Data Models in GRAPPLE framework

For each component of the GRAPPLE infrastructure, the data model is presented, showing only the innovations with respect to the previous version, described in D7.2b.

2.1 GRAPPLE Event Bus (GEB)

The GRAPPLE Event BUS (GEB) web-service uses SOAP messages for communication. For more information on SOAP, we refer to http://www.w3.org/TR/soap/

The GEB has four service methods, three for administration services and one for sending actual events. For administration these are the gebRegisterListenerOperation, the gebUnsubscribeListeneroperation and the gebListMethodsListenerOperation. For sending events there is the eventGEBListenerOperation.

The WSDL description of the below described methods is listed in the appendix of the deliverable D7.1c.

2.1.1 gebRegisterListenerOperation

The gebRegisterListenerOperation is used to subscribe a listener to GEB. An example SOAP message for registration of a webservice to GEB is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:geb="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:reg="http://xml.netbeans.org/schema/registerEventListenerRequestMsg">
   <soapenv:Header/>
   <soapenv:Body>
      <geb:gebRegisterListenerOperation>
         <registerEventListenerRequestMsg>
            <reg:eventListenerID> http://pcwin530.win.tue.nl:8080/GRAPPLE-umf-
event-bus-listener/eventEventListenerService?wsdl </reg:eventListenerID>
            <reg:methods>
               <reg:method>setUMData</reg:method>
               <reg:description>Storing statements in GUMF</reg:description>
            </reg:methods>
            <reg:methods>
               <reg:method> queryUMData</reg:method>
               <reg:description> Querying GUMF for statements</reg:description>
            </reg:methods>
         </registerEventListenerRequestMsg>
      </geb:gebRegisterListenerOperation>
   </soapenv:Body>
</soapenv:Envelope>
```

This soap message registers the webservice with the URL as listed in eventListenerID and it subscribes to two methods, namely setUMData and queryUMData. This means that whenever some server sends an event to GEB for either of the methods setUMData or queryUMData this event will be forwarded to this registered service.

The response of GEB is either true, i.e. in the following form:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns4:gebRegisterListenerOperationResponse
xmlns:ns2="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:ns3="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:ns4="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:ns5="http://xml.netbeans.org/schema/listMethodsResponseMsg">
         <registerEventListenerResponse>
            <ns2:responseMsg>true</ns2:responseMsg>
         </registerEventListenerResponse>
      </ns4:gebRegisterListenerOperationResponse>
   </S:Body>
</S:Envelope>
```

or an error message that explains why the registration did not succeed. For example, when a service is registered to the GEB, it will test if the URL used for registration is actually available and if its WSDL definition is a valid Webservice description. The error message if the URL would not be available would be something along the lines of:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
```

```
   <S:Body>
      <ns4:gebRegisterListenerOperationResponse
xmlns:ns2="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:ns3="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:ns4="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:ns5="http://xml.netbeans.org/schema/listMethodsResponseMsg">
         <registerEventListenerResponse>
            <ns2:responseMsg>Failed: javax.xml.ws.WebServiceException: Failed to
access the WSDL at:
http://pcwin530.win.tue.nl:8080/GRAPPLEDummyListener2/eventEventListenerService?ws
dl. It failed with: pcwin530.win.tue.nl.</ns2:responseMsg>
         </registerEventListenerResponse>
      </ns4:gebRegisterListenerOperationResponse>
   </S:Body>
</S:Envelope>
```

2.1.2 gebUnsubscribeListeneroperation

Unsubscribing listeners from the GEB is done using the gebUnsubscribeListeneroperation method. The unsubscribe soap message has the following pattern:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:geb="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:reg="http://xml.netbeans.org/schema/registerEventListenerRequestMsg">
   <soapenv:Header/>
   <soapenv:Body>
      <geb:gebUnsubscribeListenerOperation>
         <unsubscribeEventListenerRequestMsg>
                  <reg:eventListenerID>
         http://pcwin530.win.tue.nl:8080/GRAPPLE-umf-event-bus-
         listener/eventEventListenerService?wsdl </reg:eventListenerID>
         </unsubscribeEventListenerRequestMsg>
      </geb:gebUnsubscribeListenerOperation>
   </soapenv:Body>
</soapenv:Envelope>
```

The message only requires the ID (URL) of the listener in the reg:eventListenerID field and will delete every method associated to that ID.

The response to this message will then be (if successful):

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns4:gebUnsubscribeListenerOperationResponse
xmlns:ns2="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:ns3="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:ns4="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:ns5="http://xml.netbeans.org/schema/listMethodsResponseMsg">
         <unsubscribeEventListenerResponse>
```

```
            <ns2:responseMsg>Success, deleted
http://pcwin530.win.tue.nl:8080/GRAPPLE-umf-event-bus-
listener/eventEventListenerService?wsdl </ns2:responseMsg>
         </unsubscribeEventListenerResponse>
      </ns4:gebUnsubscribeListenerOperationResponse>
   </S:Body>
</S:Envelope>
```

2.1.3 gebListMethodsListenerOperation

The gebListMethodsListenerOperation is a simple method to list all methods listeners that are currently subscribed to the GEB. The SOAP message for this operation is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:geb="http://j2ee.netbeans.org/wsdl/GEB/gebListener">
   <soapenv:Header/>
   <soapenv:Body>
      <geb:gebListMethodsListenerOperation/>
   </soapenv:Body>
</soapenv:Envelope>
```

The response to this method is the complete lists of subscribed methods and has the following pattern:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns4:gebListMethodsListenerOperationResponse
xmlns:ns2="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:ns3="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:ns4="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:ns5="http://xml.netbeans.org/schema/listMethodsResponseMsg">
         <listMethodsResponse>
            <ns5:methods>
               <ns5:method>getCourseCount</ns5:method>
               <ns5:description>requests the number of courses currently in the
GALE instance</ns5:description>
            </ns5:methods>
            <ns5:methods>
               <ns5:method>getCourses</ns5:method>
              <ns5:description>returns the set of course, course descriptions
pairs as in the current GALE instance. <ns5:description/>
            </ns5:methods>
         </listMethodsResponse>
      </ns4:gebListMethodsListenerOperationResponse>
   </S:Body>
</S:Envelope>
```

2.1.4 eventGEBListenerOperation

The actual events are sent with the eventGEBListenerOperation. An event has the following structure in SOAP:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:even="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
xmlns:ent="http://xml.netbeans.org/schema/entListenerDefinition">
   <soapenv:Header/>
   <soapenv:Body>
      <even:eventGEBListenerOperation>
         <part1>
            <ent:method>getCourses</ent:method>
            <ent:body></ent:body>
         </part1>
      </even:eventGEBListenerOperation>
   </soapenv:Body>
</soapenv:Envelope>
```

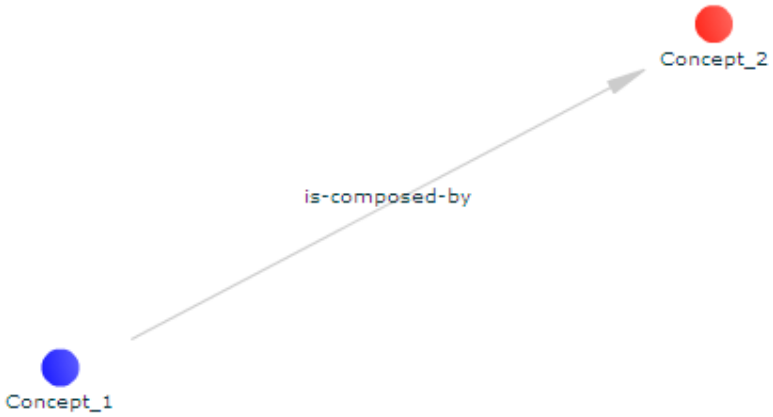
In this case the event requests the getCourses method. This method is implemented by the GALE listener. On sending the event, GEB will report a success or failure to the sender of the message. If the event has arrived well, GEB will send back an idAssignedEvent number with which responses can be tracked. Such a response will have the pattern:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:eventGEBListenerOperationResponse
xmlns:ns2="http://xml.netbeans.org/schema/entListenerDefinition"
xmlns:ns3="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
xmlns:ns4="http://xml.netbeans.org/schema/eventResponseMsg">
         <part1>
            <ns4:idAssignedEvent>283</ns4:idAssignedEvent>
         </part1>
      </ns3:eventGEBListenerOperationResponse>
   </S:Body>
</S:Envelope>
```

2.2 Domain Model (DM)

2.2.1 DM Visual

From a visual point of view the domain model is represented in a very simple way. Each concept is represented by a coloured circle and relationships between object are represented by an arrow as shown in the picture.



The relationship can be indicated to be 1-1 relationship.

2.2.2 DM Specification

The specific domain model definition is embedded inside the common part of a GRAPPLE model

```
<model>
  <header>
    // meta-data of the DM model, common to other model.
  </header>

  <body>
    <dm>
      // the DM definition
    </dm>
  </body>
</model>
```

The DM is based on the IMS VDEX specification with some extensions. We can say that a Domain model is a set of concepts (called “object” in the VDEX specification) and relationships (called “relationship object” in the VDEX specification) between concepts.

The term “object” needs to be extended with respect to the VDEX spec to fit the requirement for the definition of “concept”.

```
<term x="196" y="261">
  <termIdentifier>A9B45BB9-3BA2-D6A0-E8B5-224F2DA7543A</termIdentifier>
  <caption>
    <langstring language="it">Concept_1</langstring>
  </caption>
  <description>
    <langstring language="en"/>
  </description>
  <metadata>
    <concept>
      <lom>
        <general>
          <description>
            <langstring>xyz</langstring>
          </description>
```

```
            <catalogentry>
              <catalog>Property 1</catalog>
              <entry>
                <langstring>5</langstring>
              </entry>
            </catalogentry>
          </general>
        </lom>
      </concept>
      <resource id="7184BA47-CBD0-4189-18C1-224F76B750BC">
        <lom>
          <general>
            <title>
              <langstring>p2</langstring>
            </title>
            <description>
              <langstring>v2</langstring>
            </description>
          </general>
        </lom>
      </resource>
      <resource id="7184BA47-CBD0-4189-18C1-224F76B750BC">
        <lom>
          <general>
            <title>
              <langstring>p1</langstring>
            </title>
            <description>
              <langstring>v1</langstring>
            </description>
          </general>
        </lom>
      </resource>
    </metadata>
    <mediaDescriptor>
      <mediaLocator>http://someresource.url</mediaLocator>
      <interpretationNote>
        <langstring language="x-none">7184BA47-CBD0-4189-18C1-224F76B750BC</langstring>
      </interpretationNote>
    </mediaDescriptor>
  </term>
```

A **term** represents the entire concept. To maintain the position in the graph we added in this tag two coordinates attributes (*x*, *y*).

Inside the **term** tag:

- the *termIdentifier* identifies unequivocally the concept with a GUID,
- the *caption* contains the name of the concept,
- the *description* contains a brief description of the concept,

- the *mediaDescriptor* can be a multiple instance tags and contains references to all media objects related to the concept,
- *concept* contains the metadata and properties related to the concept itself,
- *resource* contains the properties related to media associated with the concept.

The relationship is not extended with respect to the VDEX specification.

```
<relationship>
 <sourceTerm>A9B45BB9-3BA2-D6A0-E8B5-224F2DA7543A</sourceTerm>
 <targetTerm>D653C9BB-993E-F00D-0B66-22503FD4272B</targetTerm>
 <relationshipType source="http://www.GRAPPLE.org/relations.xml">is-composed-by</relationshipType>
 <metadata/>
</relationship>
```

The relationship tag contains:

1. *sourceTerm*, which contains the ID of the source concept, which must be defined in termIdentifier within a concept
2. *targetTerm*, which contains the ID of the target concept, which must be defined in termIdentifier within a concept
3. *relationshipType*: it contains the type of the relationship. In the GRAPPLE system some of them are defined by default but the user can define his or her own.

2.3 Concept Relationship Type (CRT)

In order to exchange CRTs with other GRAPPLE components a data format has to be defined which can be written by the CRT Tool and read by other components. Most importantly CRTs have to be read by the CAM Tool and the GALE.

CRTs are expressed in an XML format and follow a XML schema specification that defines the structure of the XML format. This is necessary since it ensures that the other components are prepared for all possible variants of CRTs.

For the reason of compatibility with the other model formats of the GRAPPLE authoring tools, a common wrapper format has been defined. Each model, such as CAM, DM, and CRT, has a wrapper structure, which allows for a unified storing and retrieving from and to the database on the Web. The Web service which accepts models for storing and delivers requested models from the database needs not to have knowledge of which type the models are.

The following XML code outlines the common structure of GRAPPLE authoring tool models. Each model has a header part, where meta-data of the model is located. Meta-data includes title, common description, creation and update time, author, authorisation, and the UUID of the model.

```
<model>
  <header>
    // meta-data of the CRT model
  </header>

  <body>
    <crt>
      // the CRT definition
    </crt>
  </body>
</model>
```

The CRT definition is partitioned into seven areas, where each area is specified as a more or less complex element in the XML specification. The following piece of XML code depicts this structure with the seven elements. The numbers (as comment) before each element refer to the sub-sections below.

```
<crt>
  <!-- (1) -->  <crtdialect>...</crtdialect>
  <!-- (2) -->  <comment>...</comment>
  <!-- (3) -->  <visualrepresentation>...</visualrepresentation>
  <!-- (4) -->  <crtsockets>...</crtsockets>
  <!-- (5) -->  <adaptationbehaviour>...</adaptationbehaviour>
  <!-- (6) -->  <constraints>...</constraints>
  <!-- (7) -->  <associateddmrelations>...</associateddmrelations>
</crt>
```

(1) CRT dialect

The CRT dialect indicates if this is a "normal" CRT, a virtual reality CRT (VR-CRT) or a service relationship type (SRT). VR-CRTs and SRTs are explained in deliverables D3.4 and D3.5.

```
<crtdialect>crt</crtdialect>
```

(2) Comment

The comment field contains some free-text information about the pedagogical meaning of this CRT. This is not processed by any GRAPPLE component, but only used by authors to express and understand the pedagogical meaning of the respective CRT.

```
<comment>in prerequisite CRTs source concepts are presented before target concepts</comment>
```

(3) Visual representation

The visual representation defines how a CRT should be visually represented in the CAM. It has two elements which are shape and colour. The shape element defines the shape which is drawn on the connection line between two collections of concepts (sockets). The colour element defines the colour of the connection line and the colour of the shape surrounding the concepts. This information is only interpreted by the CAM Tool.

In the following example, a CRT is represented in green colour shaped as a diamond.

```
<visualrepresentation>
  <shape>diamond</shape>
  <colour>0x00ff00</colour>
</visualrepresentation>
```

(4) CRT Sockets

The CRT sockets define the structure of a CRT. Basically a socket is a collection containing concepts. For each socket some properties can be defined, which are colour, minimum and maximum cardinality, and the name of the socket. The UUID is automatically given by the CRT Tool. The name defines the name of the socket, the UUID is the identifier which allows for referencing this socket, and the optional colour can override the colour of the CRT for this socket. Furthermore it can be defined how many concepts have to be in this socket at least and how many concepts can be in the socket at maximum.

In the following example, a socket is defined with the name 'beginner' where at least 1 concept has to be included.

```
<socket type="source">
  <uuid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</uuid>
  <colour>0x0000ff</colour>
  <mincardinality>1</mincardinality>
  <maxcardinality>*</maxcardinality>
  <name>beginner</name>
</socket>
```

There are two ways of structuring CRTs, which are *directed* and *undirected*. In the case of a directed structure, there are two sockets whereby one must have the type *source* and the other one the type *target*. Obviously, the direction goes from source to target. In the case of undirected structure, an arbitrary number of sockets with any type can occur. The following example depicts the case of a directed structure.

```
<crtsockets>
```

```
  <socket type="source">
    <name>beginner</name>
    <uuid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</uuid>
    <mincardinality>1</mincardinality>
    <maxcardinality>*</maxcardinality>
    <colour>0x00FFCC</colour>
  </socket>
  <socket type="target">
    <name>advanced</name>
    <uuid>cf5de7f5-12b5-4720-92e5-pppppppppppp</uuid>
    <mincardinality>3</mincardinality>
    <maxcardinality>3</maxcardinality>
    <colour>0x00FFCC</colour>
  </socket>
</crtsockets>
```

(5) Adaptation behaviour

The definition of the adaptation behaviour is the central part of the CRT definition. It defines in a machine-readable way the pedagogical meaning of the CRT. The definition of the adaptation behaviour is done through a piece of code written in the GRAPPLE Adaptation Learning Environment (GALE) language. The GALE code is inserted into the XML code as it is (using CDATA). The specification of the GALE code (syntax and semantics) is done in WP1. The following example shows how the GALE code is inserted. For future purposes also other types of code can be inserted in this field.

```
<code type="gale>>
  <![CDATA[
    // gale code here
  ]]>
</code>
```

An important aspect of the GALE code is the influence on the user model variables. In the GALE code it is specified how exactly user model variables are influenced. In the CRT specification all user model variables are listed which are used by the GALE code. The following example shows how the user model variables are included in the XML specification.

```
<adaptationbehaviour>
  <usermodel>
    <umvariable>
      // specification of the first user model variable
    </umvariable>
    <umvariable>
      // specification of the second user model variable
    </umvariable>
  <usermodel>
  <code type="gale">
    <![CDATA[
      // gale code here
    ]]>
  </code>
<adaptationbehaviour>
```

The user model (UM) variables are specified in detail in the CRT code and some information is provided on each variable. The name element is used to specify the name of the UM variable. Two Boolean elements define how the variable is treated by GALE. The *public* element defines whether GALE needs to submit updates to GUMF or not. The *persistent* element defines whether the value is stored or computed or retrieved. In order to define the values used for this UM variable there is a type field specifying the value type, a default field, and a range element. The range element is used to specify the value range which, however, depends on the selected type. The default value also has to fit to the type. The following example shows the UM variable *knowledge*.

```
<umvariable>
  <umvarname>knowledge</umvarname>
  <public>true</public>
  <persistent>true</persistent>
  <type>float</type>
  <range>
     <from>0</from>
     <to>1</to>
  </range>
  <default>0</default>
</umvariable>
```

(6) Constraints

In this section three types of restrictions can be defined regarding CRTs. First, it can be specified if loops in the CAM are allowed with this CRT. For example, it would not make sense to allow loops with a prerequisite CRT. Second, it can be restricted which concepts can be put into a socket. Therefore, a tipple of socket UUID, attribute name, and attribute value is used to define that only concepts can be put into the socket with the given UUID which have set a specific attribute value for the attribute name. Third, it can be restricted which CRTs are not allowed in the "neighbourhood" when CRTs are used in the CAM. Concepts, respectively sockets containing concepts, are connected by CRT instances in the CAM. However, with the previously mentioned restriction it can be excluded that certain CRT instances are connected to the same socket. This is done by enumerating user model variables that appear in CRTs and are not allowed to be in the "neighbourhood". The following example shows how these restrictions can be specified.

```
<constraints>
  <allowedinloop>false</allowedinloop>
  <attributeconstraints>
    <attrconstraint>
      <socketid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</socketid>
      <attributename>language</attributename>
      <requiredvalue>en</requiredvalue>
    </attrconstraint>
  </attributeconstraints>
  <umvariableconstraints>
    <umvariablename>visited</umvariablename>
    <umvariablename>knowledge</umvariablename>
  </umvariableconstraints>
</constraints>
```

(7) Associated domain model relations

Associations to relationships between concepts used in the domain model can be exploited. When inserting CRTs we can lay them along existing structures of the domain model. The following example shows how relations between concepts in a domain model are used for this CRT.

```
<associateddmrelations>
  <relationshiptype>is_a</relationshiptype>
  <relationshiptype>part_of</relationshiptype>
</associateddmrelations>
```

2.4 Conceptual Adaptation Model (CAM)

The data structure of the Conceptual Adaptation Model is defined in the deliverable D3.3c. In this section we describe the CAM languages as they are implemented in GAT.

2.4.1 CAM Visual Language

The CAM visual language expresses the CAM in a visual way, and is the only language that is intended for non-programmer authors to work with. The language uses the concept of graphs to express concepts and relationships between concepts, using nodes and links. Specifically, the nodes are sockets (which contain sets of concepts from a DM), and the links are relationships from a CRT.

To summarise, we have 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

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

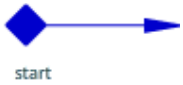


Figure 2: A CRT with one hook location.

- Groups (visually shown as sockets) of concepts of placeholders. The default representation is:

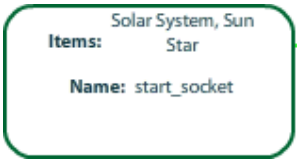


Figure 3: Grouping of concepts.

2.4.2 CAM Internal Language

The CAM internal language consists of the following parts:

- A representation of the purely visual aspects to ensure that the model appears in a consistent manner when the CAM is closed and reopened. The XML schema of the CAM
- A format that represents the CAM instance, containing links to all DMs and CRTs used within the CAM, and details of how they are instantiated.

The full schema of the CAM internal language can be found in the deliverable D3.3c.

2.4.3 CAM External Language

The CAM external language is a scaled down version of the CAM Internal Language, containing only the information required for delivery. Therefore, it contains no information about the position of CRTs or sockets within the CAM.

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 complete description of this language is included in D7.2b [19] and in [4].

2.5 User Model (UM)

The core data model of the UM is specified within the GRAPPLE Core ontology[1] (Namespace: http://www.GRAPPLE-project.org/GRAPPLE-core/, Namespace abbreviation: gc). GRAPPLE Core defines the so-called GRAPPLE statements which are the user information fragments that are stored in GUMF. GRAPPLE statements allow for arbitrary statements about users (students) and basically conform to a user-attribute-value-level structure meaning that some *user* (gc:user or gc:subject) has an *attribute* (gc:predicate) that is filled with a certain *value* (gc:object) where the degree to which the statement holds is specified via the *level* (gc:level). These statements are enriched with metadata such as the *creator* of the statement (gc:creator), the creation time (gc:created) or information regarding the origin of the statement (gc:origin). Further details on GRAPPLE statements are described in Deliverable 2.1.

[1] http://pcwin530.win.tue.nl:8080/GRAPPLE-umf/rdf/GRAPPLE-core.owl

WP2 further developed extensions for the GRAPPLE Core ontology that increase the expressivity of GRAPPLE statements. These extensions were to some extent required in order to connect GVIS, GAT and GUMF more convenient. The following table lists important language extensions provided by the GRAPPLE Extended vocabulary[2] (Namespace: http://www.GRAPPLE-project.org/GRAPPLE-extended/, Namespace abbreviation: ge).

All GRAPPLE Core extensions listed in the table are properties that can be attached to GRAPPLE statements (gc:Statement), i.e. the domain of all properties is gc:Statement. The range (allowed values) are not further restricted.

extension	description
ge:context	The context allows to specify a context in which a GRAPPLE statement was created or is valid.
ge:levelRange	Some special range definition for the gc:level: GRAPPLE statements (extended vocabulary) allow for adding a ge:levelRange that further restricts the range of a level.
ge:levelRangeMax	Some special range definition for the gc:level: the maximum value the level can be set to: ge:levelRangeMax
ge:levelRangeMin	Some special range definition for the gc:level: the minimum value the level can be set to: ge:levelRangeMin
ge:levelRangeThreshold	Some special range definition for the gc:level: some threshold value for the level, e.g. the threshold could describe the minimum level a student needs to pass a test/quiz: ge:levelRangeThreshold
ge:objectRange	Some special range definition: GRAPPLE statements (extended vocabulary) allow for adding a ge:objectRange that further restricts the range of an object value
ge:temporalStart	This property extends gc:temporal (GRAPPLECoreVocabulary.temporal) and denotes the start of some temporal period: temporalStart
ge:temporalEnd	This property extends gc:temporal (GRAPPLECoreVocabulary.temporal) and denotes the end of some temporal period: temporalEnd

Table 1 - GRAPPLE Extended ontology: additional metadata that can be attached to GRAPPLE statements.

Learning Management Systems are sending IMS-LIP-formatted information to GUMF to report about student enrolments or the result a student achieved in some test/quiz. WP2 developed a mapping tool that transforms XML-formatted IMS-LIP data into GRAPPLE statements and therefore defined an RDF vocabulary that models the IMS-LIP concepts[3] (Namespace: http://www.GRAPPLE-project.org/ims-lip/, Namespace abbreviation: ims). The main concepts of this vocabulary are listed in Table 2. The usage column explains how the vocabulary concept is applied. We differentiate between (i) properties that will be used as value of gc:predicate (*value of gc:predicate*) and (ii) properties that can simply be attached to a GRAPPLE statement, i.e. domain of the property is gc:Statement while the range is usually some string (*gc:Statement metadata*).

property (UM variable)	Description	usage
accessed	User accessed a course (or some other resource): accessed	value of gc:predicate
attends	User attends a course (the value of this property should be the ID of the course the user attends)	value of gc:predicate
changedRoleTo	User changed role (property): Property that describes the role of the student in context of a specific context.	value of gc:predicate
completedTest	User completed/finished a test or quiz	value of gc:predicate
numberOfAttempts	Number of attempts a user required for/within a test or quiz.	gc:Statement metadata or value of gc:predicate
courseId	ID of a course	gc:Statement metadata
courseTitle	name/title of a course	gc:Statement metadata
descriptionLong	Long description (<description><long> long description	gc:Statement metadata

[2] http://pcwin530.win.tue.nl:8080/GRAPPLE-umf/rdf/GRAPPLE-extended.owl

[3] http://pcwin530.win.tue.nl:8080/GRAPPLE-umf/rdf/imslip-rdf.owl

	</long></description>))	
descriptionShort	Short description (<description><short> short description </short></description>)	gc:Statement metadata
addedLearningActivity	user added a learning activity	value of gc:predicate
changedLearningActivity	user changed a learning activity	value of gc:predicate
removedLearningActivity	a learning activity was removed for the user (e.g., by the user)	value of gc:predicate
activityId	ID of a learning activity	gc:Statement metadata
activityTitle	title/name of a learning activity. This property is a sub-property of ims:title.	gc:Statement metadata
courseId	ID of a course	gc:Statement metadata
courseTitle	title/name of a course. This property is a sub-property of ims:title.	gc:Statement metadata
title	The title of something (e.g., the title of a course). This property extends dc:title.	gc:Statement metadata or value of gc:predicate
topic	The topic of something (e.g., the topic of a course).	gc:Statement metadata or value of gc:predicate
studentEnrollment	This property states that a student enrolled to a course, e.g. the gc:subject refers to the student/user and the gc:object refers to the course.	value of gc:predicate
name	the (full) name of a user	value of gc:predicate
firstName	the first/given name of a user	value of gc:predicate
lastName	the last name of a user	value of gc:predicate
email	the email of a user	value of gc:predicate
phone	the phone of a user	value of gc:predicate
gender	the gender of a user	value of gc:predicate
birthDate	the birth date of a user	value of gc:predicate
placeOfBirth	the place where the user was born	value of gc:predicate
ipAddress	the IP address of a user	value of gc:predicate
language	the preferred language of a user	value of gc:predicate
streetName	some postal address (of the user): street name	value of gc:predicate
streetNumber	some postal address (of the user): street number	value of gc:predicate
region	some postal address (of the user): region	value of gc:predicate
city	some postal address (of the user): city	value of gc:predicate
country	some postal address (of the user): country	value of gc:predicate
postcode	some postal address (of the user): postcode	value of gc:predicate
pobox	some postal address (of the user): PO Box	value of gc:predicate
organization	the organization a user is associated with	value of gc:predicate

Table 2 - IMS-LIP RDF vocabulary

The subsequent sections list examples that illustrate how the conversion between IMS-LIP data – sent by the LMS – and GRAPPLE statements – that will be stored in GUMF – is done. Further examples are available via the GUMF help pages[4].

2.5.1 Student Enrollment

[4] http://pcwin530.win.tue.nl:8080/GRAPPLE-umf/help/ims-lip-rdf.html

IMS-LIP data that describes that a student/user enrolled to a course is transformed into a single GRAPPLE statement using the ims:studentEnrollment.

IMS-LIP:

```
<?xml version="1.0" encoding="UTF-8"?>
<statement>
<origin>
<learnerinformation xml:lang="en"
xmlns="http://www.imsglobal.org/xsd/imslip_v1p0">
  <comment xml:lang="en">Student enrollment</comment>
  <securitykey>
    <keyfields>
      <fieldlabel>
        <typename>
          <tyvalue xml:lang="en">User Id</tyvalue>
        </typename>
      </fieldlabel>
      <fielddata>aclixlearn</fielddata>
    </keyfields>
  </securitykey>
  <activity>
    <contentype>
      <referential>
        <sourcedid>
          <source>LMS-CLIX-ID</source>
          <id>117359</id>
        </sourcedid>
      </referential>
    </contentype>
  </activity>
  <identification>
    <ext_identification>Learner</ext_identification>
  </identification>
</learnerinformation>
</origin>
</statement>
```

Corresponding GRAPPLE statement:

```
<gc:Statement rdf:about="http://www.GRAPPLE-project.org/umf/10505">
  <gc:subject>aclixlearn</gc:subject>
  <gc:predicate
    rdf:resource="http://www.GRAPPLE-project.org/ims-lip/studentEnrollment"/>
  <gc:object rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
      117359
  </gc:object>
  <gc:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2010-03-10T16:23:04.243+01:00</gc:created>
   <gc:creator>CLIX</gc:creator>
</gc:Statement>
```

Important parts of the corresponding GRAPPLE statement:

- gc:subject: the user who was enrolled for some course
- gc:predicate: studentEnrollment denotes that the user enrolled for some course
- gc:object: the course ID (we only take the id from the sourcedid not the source)
- ge:context / gc:spatial: if available then the ge:context and gc:spatial attributes describe in which context the user enrolled - usually the URL of the LMS instance (in the above example no context was available in the IMS-LIP data)

2.5.2 Role change

The user's role is changed: ims:changedRoleTo.

IMS-LIP:

```
<?xml version="1.0" encoding="UTF-8"?>
<statement>
       <origin>
              <learnerinformation
                     xmlns="http://www.imsglobal.org/xsd/imslip_v1p0"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:schemaLocation="http://www.imsglobal.org/xsd/imslip_v1p0
http://www.imsglobal.org/xsd/imslip_v1p0.xsd">
                     <comment xml:lang="en">Role change</comment>
                     <contentype>
                            <referential>
                                   <sourcedid>
                                          <source>ELEX</source>
                                          <id>http://localhost:3916/www/</id>
                                   </sourcedid>
                            </referential>
                     </contentype>
                     <securitykey>
                            <keyfields>
                                   <fieldlabel>
                                          <typename>
                                                 <tyvalue>User Id</tyvalue>
                                          </typename>
                                   </fieldlabel>
                                   <fielddata>student3</fielddata>
                            </keyfields>
                     </securitykey>
                     <identification>
                            <comment>Role</comment>
                            <ext_identification>Learner</ext_identification>
                     </identification>
                     <identification>
                            <comment>Role</comment>
                            <ext_identification>User</ext_identification>
                     </identification>
              </learnerinformation>
       </origin>
</statement>
```

The IMS-LIP is transformed into two GRAPPLE statements (one statement for each role):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns:gc="http://www.GRAPPLE-project.org/GRAPPLE-core/"
       xmlns:ge="http://www.GRAPPLE-project.org/GRAPPLE-extended/"
       xmlns:ims="http://www.GRAPPLE-project.org/ims-lip/"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<gc:Statement rdf:about="http://www.GRAPPLE-project.org/GRAPPLE-core/id-not-set-
yet-0">
  <gc:subject>student3</gc:subject>
  <gc:predicate
     rdf:resource="http://www.GRAPPLE-project.org/ims-lip/changedRoleTo"/>
  <gc:object>Learner</gc:object>
  <gc:creator>ELEX</gc:creator>
  <gc:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2010-03-11T08:12:50.100+01:00
  </gc:created>
  <gc:spatial rdf:resource="http://localhost:3916/www/"/>
```

```
  <ge:context rdf:resource="http://localhost:3916/www/"/>
</gc:Statement>

<gc:Statement rdf:about="http://www.GRAPPLE-project.org/GRAPPLE-core/id-not-set-
yet-1">
  <gc:subject>student3</gc:subject>
  <gc:predicate
    rdf:resource="http://www.GRAPPLE-project.org/ims-lip/changedRoleTo"/>
  <gc:object>User</gc:object>
  <gc:creator>ELEX</gc:creator>
  <gc:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2010-03-11T08:12:50.147+01:00
  </gc:created>
  <gc:spatial rdf:resource="http://localhost:3916/www/"/>
  <ge:context rdf:resource="http://localhost:3916/www/"/>
</rdf:Description>
```

Important parts of the corresponding GRAPPLE statements:

- gc:subject: the user for whom the role changes in some context
- gc:predicate: changedRoleTo denotes that the user changed his or her role to...
- gc:object: the actual role of the user
- ge:context / gc:spatial: describes in which context the user has the given role (here: http://localhost:3916/www/)

2.5.3 Tests/quizzes

A student/user completed a test/quiz: *ims:completedTest*.

IMS-LIP:

```
<?xml version="1.0" encoding="UTF-8"?>
<learnerinformation xml:lang="en"
xmlns="http://www.imsglobal.org/xsd/imslip_v1p0">
  <comment xml:lang="en">Tests/quizzes</comment>
  <securitykey>
    <keyfields>
      <fieldlabel>
        <typename>
          <tyvalue xml:lang="en">UserName</tyvalue>
        </typename>
      </fieldlabel>
      <fielddata>116004</fielddata>
    </keyfields>
  </securitykey>
  <activity>
    <contentype>
      <referential>
        <sourcedid>
          <source>LMS-CLIX-ID</source>
          <id>117359</id>
        </sourcedid>
      </referential>
    </contentype>
    <evaluation>
      <date>
        <typename>
          <tysource sourcetype="list">
              AccessDate,CreationDate,StartDate,StopDate,BirthDate</tysource>
          <tyvalue xml:lang="en">StartDate</tyvalue>
        </typename>
        <datetime>2009-10-14T10:07:47.774+02:00</datetime>
      </date>
      <date>
```

```
        <typename>
          <tysource sourcetype="list">
              AccessDate,CreationDate,StartDate,StopDate,BirthDate</tysource>
          <tyvalue xml:lang="en">StopDate</tyvalue>
        </typename>
        <datetime>2009-10-14T10:07:55.430+02:00</datetime>
      </date>
      <noofattempts>8</noofattempts>
      <result>
        <score>
          <fieldlabel>
            <typename>
              <tyvalue xml:lang="en">Total</tyvalue>
            </typename>
          </fieldlabel>
          <fielddata>30.0</fielddata>
        </score>
        <interpretscore>
          <fieldlabel>
            <typename>
              <tyvalue xml:lang="en">MinScore</tyvalue>
            </typename>
          </fieldlabel>
          <fielddata>0</fielddata>
        </interpretscore>
        <interpretscore>
          <fieldlabel>
            <typename>
              <tyvalue xml:lang="en">MaxScore</tyvalue>
            </typename>
          </fieldlabel>
          <fielddata>30.0</fielddata>
        </interpretscore>
        <interpretscore>
          <fieldlabel>
            <typename>
              <tyvalue xml:lang="en">Treshold</tyvalue>
            </typename>
          </fieldlabel>
          <fielddata>60.0%</fielddata>
        </interpretscore>
      </result>
      <description>
        <short xml:lang="en">We want to learn something about the Force</short>
      </description>
    </evaluation>
    <description>
      <short xml:lang="en">Final test for becoming a Jedi Knight.</short>
      <long xml:lang="en">star_wars jedi_knight the_force</long>
    </description>
  </activity>
</learnerinformation>
```

Corresponding GRAPPLE statement:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns:gc="http://www.GRAPPLE-project.org/GRAPPLE-core/"
       xmlns:ge="http://www.GRAPPLE-project.org/GRAPPLE-extended/"
       xmlns:ims="http://www.GRAPPLE-project.org/ims-lip/"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<gc:Statement  rdf:about="http://www.GRAPPLE-project.org/GRAPPLE-core/id-not-set-
yet-0">
  <gc:subject rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    116004
  </gc:subject>
  <gc:predicate
    rdf:resource="http://www.GRAPPLE-project.org/ims-lip/completedTest"/>
```

```
  <gc:object rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    117359
  </gc:object>
  <gc:level rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    30.0
  </gc:level>
  <ims:numberOfAttempts rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    8
  </ims:numberOfAttempts>
  <gc:creator>CLIX</gc:creator>
  <gc:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2010-03-15T10:48:31.726+01:00
  </gc:created>
  <ge:levelRange>0-30.0</ge:levelRange>
  <ge:levelRangeMin rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    0
  </ge:levelRangeMin>
  <ge:levelRangeMax rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
   30.0
  </ge:levelRangeMax>
  <ge:levelRangeThreshold>60.0%</ge:levelRangeThreshold>
  <ims:title>We want to learn something about the Force</ims:title>
  <ge:temporalStart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2009-10-14T10:07:47.774+02:00
  </ge:temporalStart>
  <ge:temporalEnd rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2009-10-14T10:07:55.430+02:00
  </ge:temporalEnd>
</gc:Statement>
```

Important parts of the corresponding GRAPPLE statement:

- gc:subject: the user who finished/completed the test or quiz
- gc:predicate: completedTest denotes that the user completed a test of quiz
- gc:object: the ID of the test or some ID that enables people to deduce which test/quiz the user completed (only the id from the sourcedid is taken, not the source)
- gc:level: the score the student/user obtained in the test or quiz
- ims:numberOfAttempts: the number of attempts the user required in/for the test or quiz
- ge:levelRange: the range of the level (gc:level) as string
- ge:levelRangeMin: the minimum value the level (gc:level) could have
- ge:levelRangeMax: the maximum value the level (gc:level) could have
- ge:levelRangeThreshold: some threshold level (gc:level), e.g. the level one requires to pass the test or quiz successfully
- ge:temporalStart and ge:temporalEnd: when the test or quiz was started/finished
- ims:title: the title of the test/quiz

2.6 GALE code (GAL)

The CRT tool allows any type of code to be attached to a particular CRT. This code will be included in the CAM that is sent to the adaptation engine. We aimed at creating a new language called GAL that should bridge the gap between authoring adaptivity and the different implementations of adaptation engines. However, at the time it seems unfeasible to create an implementation for the GAL language. When using GALE as the adaptation engine, you can use special GALE code in the CRT (called GAM code within

GALE). This code is interpreted during the compilation of a CAM to GALE. The format of GAM code is described here.

GAM code relies on GALE being the actual adaptation engine. It thus inherits GALE's view on domain-, adaptation- and user-model. In short this boils down to:

- an application is a list of concepts (there is no difference in DM and AM concepts)
- a concept has attributes and properties
- an attribute has a type (any Java type is possible), a default value or expression, event code that is executed whenever the value is changed and a set of properties
- a property has a name and a value (of type string)
- a concept can have any number of named, directed relationships to other concepts, forming a directed graph
- the user model is an overlay over the domain model (in GALE the DM and AM are merged and collectively called the domain model), where every attribute might be a user model variable.

2.6.1 GAM Code

GAM code is a list of concept definitions and their relations. A concept definition looks like:

```
<concept-uri> {
        #<attribute-name>:<type> & `<default-expr>`
        #[<persistent-attribute-name>] `<default-value>` {
                <property-name> `<value>`
        }
        <property-name> `<value>`
        <property-name> + `<value>`

        ->(<relation-name>) <relative-concept-uri> {
                <concept-definition>
        }
}
```

Some properties have a special purpose within GALE. The event property is interpreted as Java code that should be executed when the associated concept is requested by the user or when the associated attribute value is updated. Attributes can have properties like persistent, public and authoritative. The persistent property (either 'true' or 'false') is used to indicate whether updates to the corresponding UM variable should be persisted. Public, authoritative attributes are stored in GUMF, whereas public, non-authoritative attributes are requested from GUMF.

Some relations with a special purpose are the parent relation (used in hierarchical views) and the extends relation (which has a similar effect as class inheritance).

The operators '&' and '+' in the code above are used when different CRT's should be merged. They are used as operators by the compiler to merge the individual attributes and properties. If they are omitted the compiler overwrites any previous definitions.

The following example GAM code creates a concept called 'gale://gale.win.tue.nl/course/java/arrays' with one persistent attribute called 'visited'. The corresponding user model variable is incremented every time the concept is accessed. The example code:

```
gale://gale.win.tue.nl/course/java/arrays {
        #[visited]:Integer `0`
        event + `#{#visited, ${visited}+1};`
}
```

The expression syntax used inside property and attribute definitions is described in section 6.5 of D1.3.

2.6.2 Parameterized GAM

For GAM code to be of any use inside the CRT and CAM tools, we have added language construct that serve as placeholders for the actual content. A CRT has several sockets and each socket may contain any number of concepts. During compilation every combination of concepts is created and passed to the GAM code interpreter. In GAM you can refer to sockets using the %socket% notation. If a CRT has a 'source' and 'target' socket, you could write %source% and %target% to refer to these sockets.

Here is the example code for a prerequisite CRT:

```
%target% {
        #suitability:Boolean & `${%source%#knowledge > 70}`
}
```

Source is a prerequisite for target. This is done by modifying the target's default expression for the suitability attribute to include a clause that checks the user's knowledge of the source concept.

The same parameter notation can be used for custom CRT parameters.

2.7 GRAPPLE Conversion Component (GCC)

The GRAPPLE Conversion Component (GCC) has not changed since the previous version, therefore the description already documented in chapter 2.9 of D7.2b can be considered final.

3 Conclusions

The data models described here constitutes the documentation of the work carried out during the third year of the GRAPPLE project.

This deliverable should be considered as a basis for the deliverables D7.1c and D7.5, referring respectively to the final specifications and the final implementation of the overall GRAPPLE system.

References

1. Oneto L. et al: D7.1b - Updated specification of the operational infrastructure
2. Simon, B. et al, 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. et al: 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 al: D3.1 - Design specification of a DM definition tool
10. Hendrix M., et al: 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 al: D1.1b - CAM to adaptation rule translator – implementation
15. Oneto L., et al: 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.
18. D3.3c - Final implementation of the CAM definition tool.
19. D7.2b - Lucia Oneto et al. - Data models and related documentation - update