



GRAPPLE

D7.1b Version: 2.0

Updated specification of the operational infrastructure

Document Type	Deliverable
Editor(s):	Lucia Oneto (Gilabs)
Author(s):	Lucia Oneto (Gilabs), Kees van der Sluijs (TUE), Jose Luis Santos Odriozola (Atos), Maurice Hendrix (Warwick), Alexandra Cristea (Warwick), Michele Dicerto (Gilabs), Eelco Herder (I3s), Fabian Abel (I3s), Luca Mazzola (USI), David Smits (TUE), Ekaterina Pechenezhskaya (TUE), Fabrizio Anagi (Gilabs)
Reviewer(s):	Christian Glahn (OUNL), Alexander Nussbaumer (UniGraz), Cord Hockemeyer (UniGraz)
Work Package:	WP7
Due Date:	M18
Version:	2.0
Version Date:	05-02-10
Total number of pages:	43

Abstract: This document presents the initial specifications of the interfaces to be designed and developed in the GRAPPLE Operational Infrastructure, where all the different components of the system will be integrated (ALE, LMS, authoring tools, user interfaces, etc).

Keyword list: Learning Management Systems, Adaptive Learning Environment, integration

Summary

In order to obtain a final system that is actually able to provide effective and efficient adaptive courses, it is necessary to identify the interfaces to be designed and developed in the GRAPPLE Operational Infrastructure, where all the different components of the system will be integrated (ALE, LMS, authoring tools, user interfaces, etc).

Authors

Person	Email	Partner code
Kees van der Sluijs	k.a.m.sluijs@tue.nl	TUE
David Smits	D.Smits@tue.nl	TUE
Ekaterina Pechenezhskaya	e.pechenezhskaya@tue.nl	TUE
Fabrizio Anagi	f.anagi@giuntilabs.com	GILABS
Lucia Oneto	l.oneto@giuntilabs.com	GILABS
Michele Dicerto	m.dicerto@giuntilabs.com	GILABS
Maurice Hendrix	maurice@dcs.warwick.ac.uk	WARWICK
Alexandra Cristea	A.I.Cristea@warwick.ac.uk	WARWICK
Eelco Herder	herder@l3s.de	L3S
Fabian Abel	abel@l3s.de	L3S
Jose Luis Santos Odriozola	jose.santos@atosresearch.eu	ATOS
Luca Mazzola	luca.mazzola@lu.unisi.ch	USI

Table of Contents

SUMMARY	2
AUTHORS	2
TABLE OF CONTENTS	2
TABLES AND FIGURES.....	4
LIST OF ACRONYMS AND ABBREVIATIONS	4
1 INTRODUCTION	7
1.1 Task and Deliverable Description	7
2 GRAPPLE OPERATIONAL INFRASTRUCTURE OVERVIEW	7
2.1 Scenarios.....	7
2.1.1 Scenario 1 – Paul and the Solar System application	8



- 2.1.2 Scenario 2 – Kees and the Space Physics application 10
- 2.1.3 Scenario 3 – Nicole and the Solar System application 10

- 3 GRAPPLE KEY COMPONENTS – SECOND PROTOTYPE..... 11**
- 3.1 Shibboleth – Single Sign On facility 13**
- 3.2 GRAPPLE Event Bus – GEB (WP7)..... 14**
 - 3.2.1 Component Functionality..... 14
- 3.3 GRAPPLE User Model Framework – GUMF (WP2 & WP6) 20**
 - 3.3.1 Component functionality..... 20
 - 3.3.2 Services offered to other components..... 21
 - 3.3.3 Services required to other components..... 22
- 3.4 GRAPPLE Adaptive Learning Engine – GALE (WP1)..... 23**
 - 3.4.1 Component Functionality..... 23
 - 3.4.2 Services offered to other components..... 23
 - 3.4.3 Services required to other components..... 24
- 3.5 GRAPPLE Authoring Tool – GAT (WP3)..... 24**
 - 3.5.1 Component Functionality..... 24
 - 3.5.2 Services offered to other components..... 26
 - 3.5.3 Services required to other components..... 29
- 3.6 GRAPPLE Visualisation – GVIS (WP4) 29**
 - 3.6.1 Services offered to other components..... 29
 - 3.6.2 Services required to other components..... 30
- 3.7 LMS 30**
 - 3.7.1 Component Functionality..... 30
 - 3.7.2 Services offered to other components..... 30
 - 3.7.3 Services required to other components..... 31

- 4 ANNEX A – SHIBBOLETH..... 31**
- 4.1 Installation of Shibboleth IdP 31**
- 4.2 Configuration of IdP 31**
 - 4.2.1 Definition of the authentication services..... 34

- 5 ANNEX B - SERVICE INTERFACE GUIDELINES & GRAPPLE WSDL FILES 35**
- 5.1 eventEventListener..... 36**
 - 5.1.1 eventDispatchedRequestMsg.xsd..... 36
 - 5.1.2 eventEventListener.wsdl..... 36
 - 5.1.3 eventEventListenerWrapper.wsdl..... 37
- 5.2 eventGEBListener..... 37**

5.2.1 eventRequestMsg.xsd 37

5.2.2 eventResponseMsg.xsd 38

5.2.3 eventGEBListener.wsdl 38

5.2.4 eventGEBListenerWrapper.wsdl 39

5.3 GebListener 40

5.3.1 registerEventListenerRequestMsg.xsd..... 40

5.3.2 registerEventListenerResponseMsg.xsd 40

5.3.3 listMethodsResponseMsg.xsd 40

5.3.4 gebListener.wsdl 41

5.3.5 gebListenerWrapper.wsdl 42

REFERENCES 43

Tables and Figures

List of Figures

Figure 1: GRAPPLE second prototype architecture. 12

Figure 2: Shibboleth and GRAPPLE. 14

Figure 3: Statechart example of the *register* service. 16

Figure 4: Statechart example of the *listMethods* service. 17

Figure 5: Sequence diagram example of using asynchronous event method over GRAPPLE Event Bus. (First part)..... 19

Figure 6: Sequence diagram example of using asynchronous event method over GRAPPLE Event Bus. (Second part)..... 20

Figure 7: GRAPPLE Authoring Tool..... 26

Figure 8: Schematic overview of authoring tool WSDL..... 26

List of Acronyms and Abbreviations

ADL	Advanced Distributed Learning
AH	Adaptive Hypermedia
AHAM	Adaptive Hypermedia Application Model
AICC	Aviation Industry CBT (Computer-Based Training) Committee
ALE	Adaptive Learning Environment
AM	Adaptation model
API	Application Programming Interface
AS	Authentication Service
AXIS	Apache eXtensible Interaction System
CAM	Conceptual Adaptation Model
CAS	Central Authentication System

CMS	Content Management System
CRT	Concept Relationship Type
CSV	Comma Separated Values
DM	Domain Model
DS	Discovery Service
EB	Event Bus
ES	Event Service
GALE	GRAPPLE Adaptive Learning Engine
GAT	GRAPPLE Authoring Tool
GCC	GRAPPLE Conversion Component
GDA	GRAPPLE Device Adaptation
GEB	GRAPPLE Event Bus
GID	GRAPPLE IDentifier
GRAPPLE	Generic Responsive Adaptive Personalized Learning Environment
GUMF	GRAPPLE User Modeling Framework
GVIS	GRAPPLE VISualization
IMS	Instructional Management System
IMS CP	IMS Content Packaging
IMS LD	IMS Learning Design
IMS QTI	IMS Question and Test Interoperability
IMS SS	IMS Simple Sequencing
JMS	Java Message Service
LAOS	Layered WWW AH Authoring Model and their corresponding Algebraic Operators
LCMS	Learning Content Management System
LDAP	Lightweight Directory Access Protocol
LMS	Learning Management System
PHP	PHP Hypertext Preprocessor
OWL	Web Ontology Language
RDFS	Resource Description Framework Schema
RP	Relying Party
SAML	Security Assertion Markup Language
SCORM	Shareable Content Object Reference Model
SOAP	Simple Object Access Protocol
SP	Service Provider
SPIP	Système de Publication pour l'Internet Participatif
SSO	Single Sign On
UM	User Model
UMF	User Model Framework
WAYF	Where Are You From

WP	Work Package
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
VLE	Virtual Learning Environment
VRE	Virtual Research Environment
XML	eXtensible Markup Language

1 Introduction

The GRAPPLE system is a component-based service oriented system: the major functions are split into modules or components, which in turn are split into services. This deliverable documents the functional architecture of the GRAPPLE system and its integration with existing LMSs in academic and industrial settings. More specifically, it describes the current separation of the whole system into independent components interfaced to external LMSs, and the services that each component offers. Furthermore, it describes the interaction between the components when executing the core functionality of the system.

This deliverable is an update of the previous deliverable, D7.1a and it needs a further update that will be done in D7.1c – Final specification of the operational infrastructure. It includes some of the decisions taken during the second prototype development work.

1.1 Task and Deliverable Description

T 7.1 Operational infrastructure specification and design (GILABS, ATOS, TUE, LUH, IMC)

This task will result in the design of the overall GRAPPLE Operational Infrastructure where all the different components of the system will be integrated (Adaptive Learning Environments, Learning Management Systems, User Model Framework, authoring tools, user interfaces, etc.). The main activities will include an analysis of the components selected/developed by other WPs, the identification of the communication channels amongst them and the design of suitable interfaces. The communication will be realized through web service architecture. Available commercial and open source LMSs will be considered and analyzed (Claroline, Sakai, Moodle, but also others, e.g. Learn eXact and IMC CLIX) in order to understand and model the underlying requirements for interoperability in existing learning environments.

D7.1a Initial specification of the operational infrastructure (GILABS, M9)

This deliverable will contain the initial specification of the interfaces to be included in the GRAPPLE operational infrastructure. This design document will be the basis for the implementation of the first prototype of the GRAPPLE system (D7.4.a).

D7.1b Updated specification of the operational infrastructure (GILABS, M18)

According to the cyclical approach proposed for the implementation and integration, the operational infrastructure specification will be incrementally updated during the project lifetime; hence this will be the first update.

D7.1c Final specification of the operational infrastructure (GILABS, M27)

This deliverable will contain the final and detailed specification of the interfaces to be included in the GRAPPLE operational infrastructure.

2 GRAPPLE Operational Infrastructure Overview

2.1 Scenarios

For presenting the core interaction between the system services a scenario-driven description approach has been selected: the chosen scenarios have been proposed for the second prototype and they involve the following GRAPPLE components:

- LMS (Learning Management System) in charge of delivering the non-adaptive courses and of providing a container for the adaptive courses.
- GALE (GRAPPLE Adaptive Learning Engine) as GRAPPLE Core Engine in charge of delivering the adaptive courses. GALE includes its own authoring component and user model.
- GAT (GRAPPLE Authoring Tool) in charge of the authoring part. It is composed by three elements, DM (Domain Model), CRT (Concept Relationship Type) and CAM (Conceptual Adaptation Model) tools.

- GUMF (GRAPPLE User Modeling Framework) provides the functionality for storing, sharing and querying user model attributes, as provided by the LMS and GALE. Customizable reasoners and mappers are available for converting user model data upon request.
- GVIS (GRAPPLE Visualization) is in charge of providing meaningful representations of the User Model data enriched with data from GALE, LMS and from the full learning environment in order to support learners during self-reflection and to offer an informative tool to tutors and teachers.
- GCC (GRAPPLE Conversion Components) provide the interoperability layer between the single existing LMS and the GRAPPLE components.
- GDA (GRAPPLE Device Adaptation) is in charge of the content adaptation in function of the device characteristics.

2.1.1 Scenario 1 – Paul and the Solar System application

Author and Teacher Paul authors an application (adaptive course) about the Solar System in GRAPPLE with the GRAPPLE authoring tools and integrates an existing quiz from an LMS, which he has authored some time ago. The LMS shows the progress of the learners.

1. LMS

At the GRAPPLE setup the LMS administrator Carl decides which variables can be available for policy reasons. By default all the variables made available by Carl are private.

Now, Paul defines which UM variables available in the LMS can be public, such as the quiz score. Any time Paul can change the UM variables status to public or private.

Paul uses the LMS in one of the LMSs to offer a quiz to the user about the Sun concepts. The quiz has 5 multiple choice questions, and the result of the test by a user will be a number between 0-5. Paul indicates that the result of the quiz is externally available under the name `sun.quizScore`.

In order to guarantee interoperability between GRAPPLE system and different LMSs, when possible, the most of the LMS variables are mapped to a common data format valid for all the GRAPPLE system. However this can be not possible for all the variables.

2. DM

Paul creates a new domain in the domain tool. It contains concepts like “sun”, “earth”, “moon”, and relationships like “hasplanet” and “hasmoon”.

3. CRT

Paul uses the CRT tool to create the simplistic `Paul_prereq` CRT that specifies that:

“in order for a user to see information about concept X, the user needs enough knowledge in the user model about concept Y and enough knowledge in the user model about concept Z”, and an instantiation of this rule will mean that “all links to a ‘page’ about concept X will get a condition (about concept Y and Z) that if the condition is true (so the condition is satisfied) these links are highlighted, otherwise the links are hidden.

In order to provide compact views in visualization of student models, Paul creates a CRT (*main-concepts-crt*) that allows establishing (via placeholders) which concepts are considered as “main concepts”. The main concepts will be included in the visualizations, while the others, considered of secondary importance, will be included on visualization only on user request. The CRT also specifies how the visualization of these concepts is to be done. In this case, it indicates the expected level of proficiency that the students have to achieve in every main concept in the placeholders. These expected levels will be used in visualizations to represent the learning objectives of students. The knowledge variables for these major concepts are declared public. Paul can use existing User Model variables in his crt, or create new ones. As he prefers using existing conventions and names, he decides to look what UM variables are already present. For this purpose the CRT tool has a simple UM query interface. The query interface can query for all variables in use in any adaptive course (that is uniquely associated to a CAM) by anyone, or be more specific. Paul could specify a specific CAM (adaptive course), or a specific concept. The query interface will list UM variables as follows:

Name, range and Used by exists in all

Name	Range (textual description of the range for the authors convenience)	Used by *(e.g. used by CAM:SolarSystem, CAM:ArtsHistory or CAM:ArtsHistory.Michelangelo)
------	--	--

*The *used by* describes where the variable is used. It is tied to a specific concept in a specific application, all concepts in a specific application, is it a general UM variable in some application, e.g. learning styles.

In this particular case, no relevant UM variables are found, but Paul will ensure that the LMS quiz results will be available at a later stage.

4. CRT

In order to indicate, the expected end state(s) of a adaptive course Paul creates a CRT (*end-of-course-crt*) that allows to establish (via placeholders) which concepts are involved in the end state and he can indicate the expected level of proficiency that the students should have reached at the end state.

5. CAM

In the CAM tool, Paul indicates that the Paul_prereq needs to be applied to the concept 'earth'. The prereq refers to the concept 'sun', with the predicates 'externalKnowledge' and 'knowledge' and a level of 100%.

This means that Paul wants to give access to a page about concept "earth" if the user has acquired internal knowledge about the Sun. In addition, Paul also wants to make sure that the user not only knows about the sun via navigating through the adaptive application, but also via filling in a test in one of the LMSs CLIX, CLAROLINE, SAKAI or learn eXact (or some other LMS). The UM variable "sun.externalKnowledge" represents having passed one of these tests in one of the LMSs: it can be mapped to a Boolean value.

In the CAM tool, Paul drags and drops the main concepts into the main-concepts-crt, as he wants to enable the visual representations of the knowledge of concepts of the adaptive course. This will ensure that the selected concepts will be used in the visualization to represent the learning objectives of the students. When students have to choose between 2 assignment topics, Paul has to indicate two slightly different possible end states. The advantage of indicating an end state is that the visualization will be able to derive and visualize the learning goals from this. In addition to this it makes it clearer for Paul where his adaptive course can end. The consistency check in the CAM will check that there are no unreachable states in it, as far as static analysis goes, so this will help Paul to make sure students can actually reach one of his intended end states and therefore achieve the learning goals.

6. GUMF mappings

Paul has created a test in the LMS called SolarSystemInterest_test. This test tests students on their interests. It aims to find out whether they are more interested in elementary particle physics, or in astronomy. This test is different from a conventional test where the result ultimately is pass or fail. Paul could write adaptation rules for this behavior directly in a CRT. However with possible sharing in mind, Paul decides to map the results of his test, which will be a number between 0 and 10, to the UM variable interest, which can either take the value "*particle_physics*" or "*astronomy*".

In order to do this, there is a GUMF mapping tool, in the GAT. He opens the GUMF mapping tool, locates the test and specifies the mapping.

7. CRT

Paul also wants to show the progress of the user in the adaptive course in the LMS. Therefore in the CRT, Paul indicates which UM variables are public, i.e. shared with the GUMF. Paul selects the knowledge attributes of the major concepts in the adaptive course.

8. GALE

Once the application has been defined in the CAM, Paul makes it available to GALE. The process will be triggered by Paul in the CAM interface.

9. LMS

Paul makes the application Solar System available from the LMS and enrolls students following the normal procedure.

10. GVIS

Paul logs on the LMS and enables a graphical widget that provides a compact indicator of the learner's progress in the Solar System application. Paul wants to investigate the adaptive course status, and review the progress of learners over adaptive course. He notices that the **overall indicator** reports interesting information on the global status of the class. When the instructor clicks on it, a different view is opened, and reaches a **detailed view** where a graphical representation of the **knowledge level**, the **expected target level**, and of the **current trend** is presented for main concepts of the adaptive course. In this way, the instructor can notice in which concepts the students are performing low and take appropriate actions.

2.1.2 Scenario 2 – Kees and the Space Physics application

Author and Teacher Kees author an application (adaptive course) about Space Physics in GRAPPLE with the GRAPPLE authoring tools. Kees knows Paul and his Solar System application. As Kees knows that some students that did Paul's adaptive course on the Solar System, will also do the Space Physics adaptive course, Kees wants to prevent that users that already know concepts that were already discussed in the Solar System adaptive course have to redo them again.

1. DM

The domain of the Space Physics application contains concepts like "planet" and "star".

Kees can access to the Solar System application and use some existing concepts such as "sun" and "earth".

2. CRT

Now Kees wants to update the knowledge for these concepts by using the information from Paul's concepts of "earth" and "sun".

Kees can get the list of UM variables already available from the GUMF and he can decide to use some of them or to create new UM variables. Then Kees can select and use some UM variables already defined and available in the GUMF (made available by Paul to the other authors in Step 7).

Kees can choose to use directly sun.knowledge variable or to create star.knowledge(ext) that will be filled by a UM mapping. Kees decides to create a rule that when sun.knowledge is larger than 80%, star.knowledge is increased with 10%.

Kees uses the CRT Kees_prereq that specifies that:

"if the learner has knowledge of the concept Y and/or Z, an instantiation of this rule will mean that "all links to a 'page' about concept X will get a condition (about concept Y and Z) that if the condition is true (so the condition is satisfied) these links are hidden, otherwise the links are highlighted".

3. CAM

Kees uses the CAM tool to indicate that the Kees_prereq CRT needs to be applied, to concept "sun" (for concept X), and UM variable "star.knowledge" as UM variable Y.

Kees can get the list of UM variables already available from the GUMF and he can decide to use some of them.

This means that Kees wants to give access to a page about concept "sun" if the user does have the internal knowledge about the star.knowledge.

2.1.3 Scenario 3 – Nicole and the Solar System application

Student Nicole has to follow the applications Solar System and Space Physics. Her UM data is available in the LMS and in the GUMF.

1. LMS

Nicole accesses to the LMS and she can see the list of courses where she is enrolled in. The list can have some presentation features adapted to her UM data provided by the GUMF and used by the CRT.

Before starting the adaptive course Solar System, she has to do a quiz in the LMS. The information related to the event "quiz results" is provided to the GUMF (with object 'sun' and predicate 'quizScore').

2. GUMF

The update of the quizScore for Nicole triggers the reasoner plugin, created by Paul. The quizScore for 'sun' is converted to externalKnowledge for 'sun'.

In detail the rule states that for: - Subject: a person/user

- Predicate: quizScore

- Object: sun

- Level: 1-5

a new statement will be generated with:

- Subject: same as above

- Predicate: externalKnowledge

- Object: sun (as above)

- Level: the above level converted to a 0-100 scale.

3. LMS

Nicole selects the adaptive course Solar System and the LMS launches the application from GALE. The information related to the event "access to a course" will be provided to the GUMF.

4. GUMF

The GUMF manages the update of UM information about Nicole.

5. GALE

GALE provides the application updated with Nicole's UM info provided by the GUMF.

6. GVIS

Nicole wants to check her progress with the adaptive course. She can see in the LMS interface a **visual indicator** (widget) that presents a simplified view of her progress with the adaptive course. Moreover, other compact indicators are provided: the overall level of knowledge, compared to the overall level of the class, and the target level defined by Paul.

Based on the information presented by this initial aggregated indicator, Nicole is able to understand her personal situation on the adaptive course and is stimulated to explore her analytic profile. The **analytic profile** presents a set of detailed view of the learner's performance data. In particular, she can see a list of the main concepts of the adaptive course. For each of these concepts, the visualization reports **the level of knowledge**, **the level of knowledge of the class**, and **the target level**. She also can compare her profile with those of other students. This will help learners to find peers who are strong in areas, in which they are weak in and vice versa. Using these graphical representations that provide a fine level of details of concepts of the adaptive course, Nicole reflects on her learning status, identifies her strengths and weaknesses, and achieves a deeper understanding of the personal knowledge.

7. LMS

When Nicole exits from the adaptive course, she is back to the list of courses where she is enrolled. The list can have some presentation features adapted to her UM data provided by the GUMF, such as the icon annotation that indicates which adaptive courses have been completed successfully or not.

3 GRAPPLE key components – second prototype

The first prototype described in D7.1a was the first step towards the GRAPPLE framework able to be integrated with existing LMSs. In the first year of the project only two modules were available: GALE, the GRAPPLE Core Engine, and the LMSs belonging to the consortium.

The second prototype involves a considerable improvement in comparison with the previous version: the other Work Packages (WPs) have developed new GRAPPLE modules and they need a solid and stable infrastructure able to integrate them.

Figure 1 displays the GRAPPLE components and the communication among them. It is possible to identify a key element, the GRAPPLE Event Bus that handles the connections among the LMS, GUMF, GALE, the

GRAPPLE Core Engine, GAT and the GVIS components, and Shibboleth as technological choice for web single sign-on across or within organizational boundaries.

GCC provides an interface for LMS to the GEB. It is therefore a wrapper that converts LMS specific functions and data structures into information that can be handled by the GRAPPLE infrastructure. This justifies its collocation in Figure 1: the single GRAPPLE Conversion Component manages and maps the LMS specific parameters to standard format (IMS LIP), as specified in D7.2b [18]. Then it is necessary to have a GCC for any LMS integrated to GRAPPLE.

GDA component is responsible of the content adaptation based on the type of device that is used by a learner to access to the adaptive course. This component is a module located in the GALE, the GRAPPLE core engine in charge of delivering the adaptive courses.

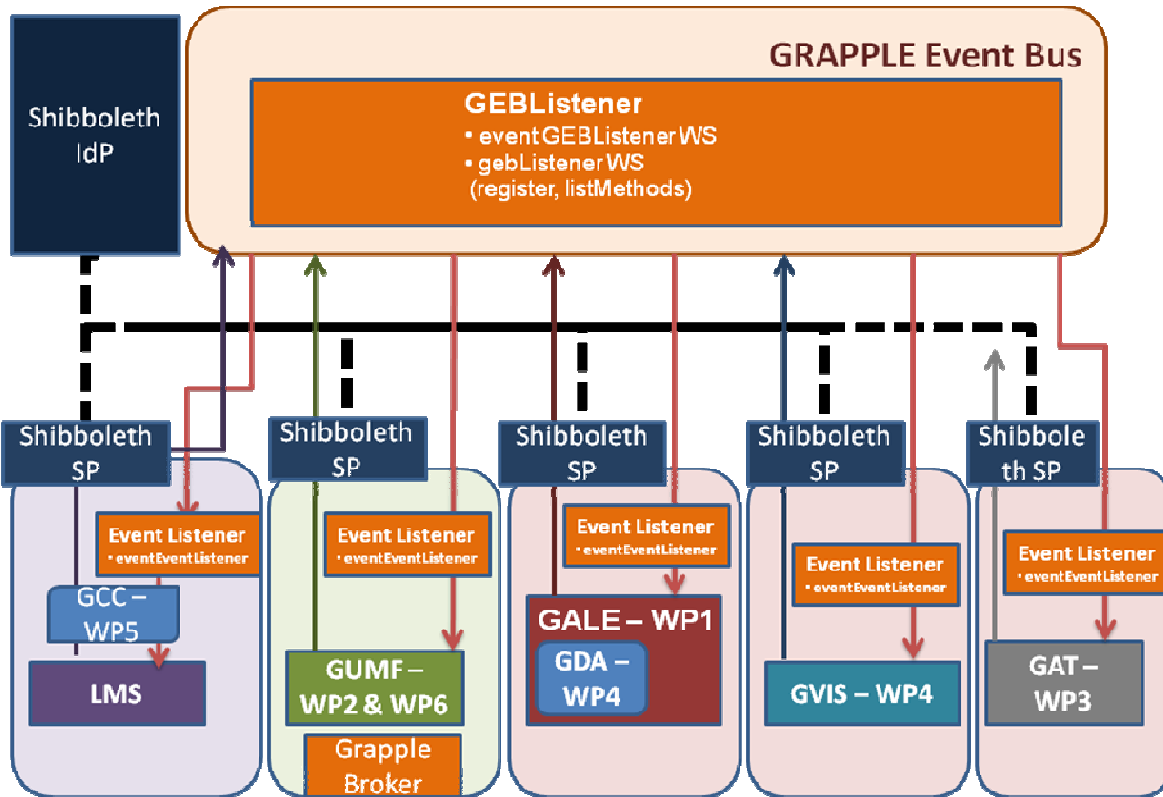


Figure 1: GRAPPLE second prototype architecture.

All components are connected through the GRAPPLE Event Bus (GEB). Each component may send requests to the GEB, where other components may listen to these events and handle them. This may result in a reply sent to/through the GRAPPLE Event Bus as well. In specific, (1) all the GRAPPLE components send just events and other ones receive these events if they listen and (2) any GRAPPLE component can send requests to the GEB and only the addressed component responds.

Figure 1 illustrates the communication among the main GRAPPLE components: even if they are not present, any component can provide a set of web services.

Each arrow represents a service provided by the target component to the source component of the arrow:

- On the Event Bus side there is the *eventGEBListener* web service that collects all the events sent by the LMS, GUMF, GVIS, GAT and GALE
- On the single GRAPPLE components side, there is the *eventListener*, with the web service *eventEventListener*, that processes the events that have been rerouted by the GRAPPLE Event Bus to all the GRAPPLE components that have registered for the specific type of event. The *eventListener* has the possibility to register in the GRAPPLE Event Bus subscribing methods used by the single GRAPPLE components to listen to the GRAPPLE Event Bus. So any time an event comes in, it is rerouted to the GRAPPLE components. By using the parameters passed by *eventListener*, it will be possible to trigger one of the web services made available by the single

GRAPPLE components, such as Grapple Broker by the GUMF, or a possible web services by the other components.

3.1 Shibboleth – Single Sign On facility

In order to allow the user to sign only once in the GRAPPLE system, it is necessary to have a Single Sign On (SSO) facility.

SSO is a mechanism through which a user can access different systems and services with a single password. SSO makes the system more usable for the authors and learners and provides easier management of access privileges to various resources. There are many SSO mechanisms and not all of them are supported by all the LMSs present in the consortium.

Shibboleth [14] [15] is a free, open-source web Single Sign-On system (SSO) based on the open standard Security Assertion Markup Language (SAML). Shibboleth supports secure access to resources across security domains. Two main components of Shibboleth are Identity Provider (IdP) and Service Provider (SP). In Figure 1 it is possible to identify in the GRAPPLE framework where the IdP and the single SPs are located.

IdP supplies information about a user to a SP which gathers information about users to protect resources. The typical use case is the following: a web browser accesses a protected resource, authenticates at their IdP, and ends up back at the resource logged in.

This procedure consists of the following four steps:

- *Access of the protected resource.* When a user tries to access a [protected resource](#) (defined in web service configuration), Shibboleth intercepts the request. Basing on the protection configuration, the SP selects a [session initiator](#) to use. A session initiator figures out which IdP the user will authenticate with and what protocols should be used. The profile preferences signal between providers through [metadata](#), which can supply a text entry box, send the user to a Discovery Service (DS), or redirect the user to a common IdP.
- *User Authentication to the IdP.* SP issues an authentication request to the IdP with the format based on the profile they choose to speak. Then, the authentication request is placed in the browser, and the user is redirected back to the right endpoint at the IdP. Basing on the request and on the configuration of IdP for this SP and authentication in general, the IdP decides how the user would be authenticated. The user is redirected off to the right login handler, authenticates through the method selected, and comes back to the profile handler with their username set.
- *Issuing of the response to SP by IdP.* The IdP collects a set of attributes for the user through the [attribute resolver](#) (which if necessary transforms the user data and attaches encoders to each attribute). The [attribute filter](#) constrains the set of information sent in the response. In order to protect the user's privacy, the set of attributes released most often depends on the SP and the principal. The user's information is all wrapped up into a message with the encoders attached earlier (in a SAML 2.0 assertion). This assertion is [signed with the IdP's key](#) and [encrypted with the SP's key](#) for security and privacy. The assertion is placed into a message and the user is redirected to the SP carrying it along.
- *Back to the SP.* The assertion consumer service at the SP unpacks the message, decrypts the assertion, and performs several security checks. If everything's in order, then it extracts attributes and other information from the message. SP's [attribute extractor](#) translates attributes into local environment or header variables. The SP can then [enforce rules itself](#) or just pass along the attributes to the application to use however it requires.

In GRAPPLE Project Shibboleth is used for single sign-on between GRAPPLE (the GRAPPLE Event Bus is the GRAPPLE component directly involved in this) and LMSs – Moodle, Sakai, Claroline, CLIX, learn eXact.

Shibboleth IdP asserts unique digital identities to the users. So it:

- connects to authentication and user data systems of GRAPPLE and LMSs,
- provides information about how a user has been authenticated,

- provides user identity information from the data source.

GRAPPLE (and LMSs) serves as SP and they:

- initiate the requests for authentication and attributes,
- process incoming authentication and attribute information.

In the following part of this section we first overview the installation of Shibboleth IdP, then we describe configuration of IdP, and, finally, we consider definition of the authentication services.

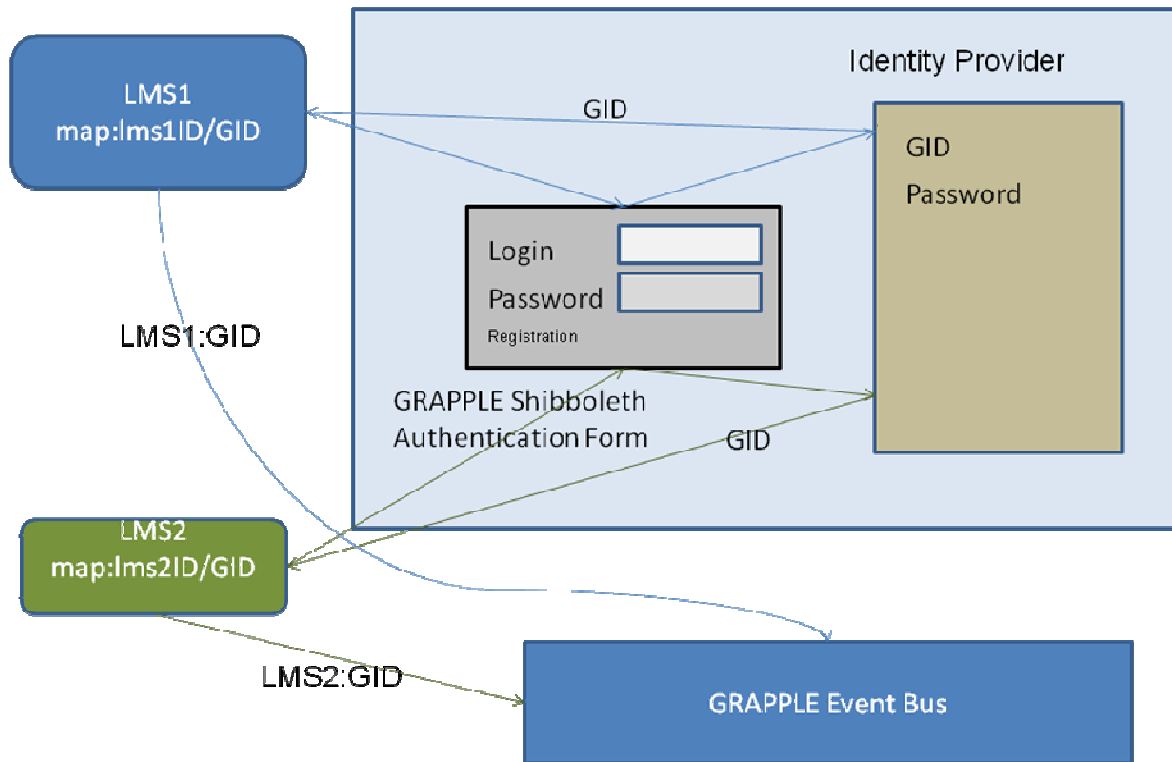


Figure 2: Shibboleth and GRAPPLE.

The GRAPPLE Identifier (GID) is decided by the user himself. The Identity Provider prompts a proper window to create his new GID.

Installation and configuration of Shibboleth IdP are described in the ANNEX A – Shibboleth.

3.2 GRAPPLE Event Bus – GEB (WP7)

3.2.1 Component Functionality

Communication within the GRAPPLE framework is facilitated by the GRAPPLE Event Bus that supports the following core functions:

- managing the valid interfaces
- facilitating the communication between the components.

The Event Bus supports a number of services that should be used by all communicating components that use the GRAPPLE Framework, i.e. the GRAPPLE Core Engine, the GUMF, the LMSs and other components.

To properly connect to the Event Bus the LMS should use the following Event Bus operations:

- *register(eventListenerID, methods):Bool* (ref to 5.3)

The *register* method is a request-response service. It means that the *eventListener* component sends a register request and it will wait for a response.

The *register* method is used to subscribe a new component's *eventListener* to the Event Bus.

- *eventListenerID*, a unique identifier (String) that contains the address of the *eventListener* where forwarded messages can be sent to.
- *methods*, a list of methods that the new *eventListener* subscribes to. Components only handle specific methods (i.e. the one that is called in an event '*method*'). The Event Bus will only send events to component that listens to that specific event. *methods* are complex type values, composed by one string describing the name of the method and a second string describing the functionality of the method; it is an attribute with cardinality of 1 to n.

The *register* method returns a Boolean, true, if the registration has been completed successfully or false in any other case.

The following Sequence diagram is an example of the user of the *register* method by the GRAPPLE User Model Framework. The component's *eventListenerID* is in this case "GUMF" and the GUMF component supports the methods *registerUMApplication*, *setUMInferenceRules*, *setUMMappingRules*, *setUMData* and *queryUMData* (these methods are detailed below).

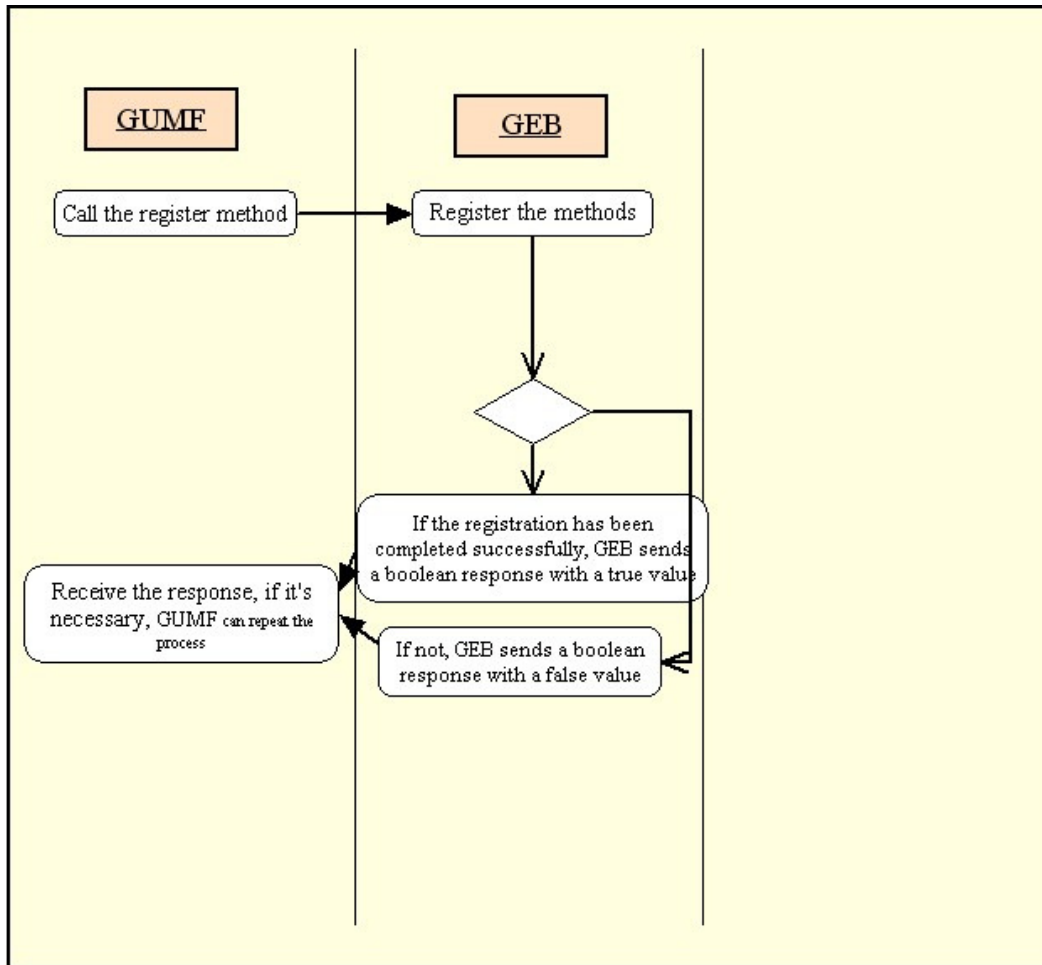


Figure 3: Statechart example of the *register* service.

- *listMethods():methodsInformation* (ref to 5.3)

The *listMethods* method is a request-response service. It means that the *eventListener* component sends a *listMethods* request and it will wait for a response.

The *listMethods* method is used to get the lists of methods registered from the all component listeners.

- It returns *methodsInformation* complex type. It is composed by a string which identifies the ID of the *eventListener* and other complex type called methods which are composed by a string which identifies the name of the method and other string which contents the description of the method.

This method has been defined for development purposes, to allow the developers to interrogate the other components.

The following Sequence diagram is an example of the user of the *listMethods* method by a request component, one analyst needs to know which kind of information can get from the GRAPPLE environment. In this case, the user makes the request to GEB and only obtains the methods registered from GUMF, because actually it is the unique component which has registered methods. The methods are *registerUMApplication*, *setUMInferenceRules*, *setUMMappingRules*, *setUMData* and *queryUMData*.

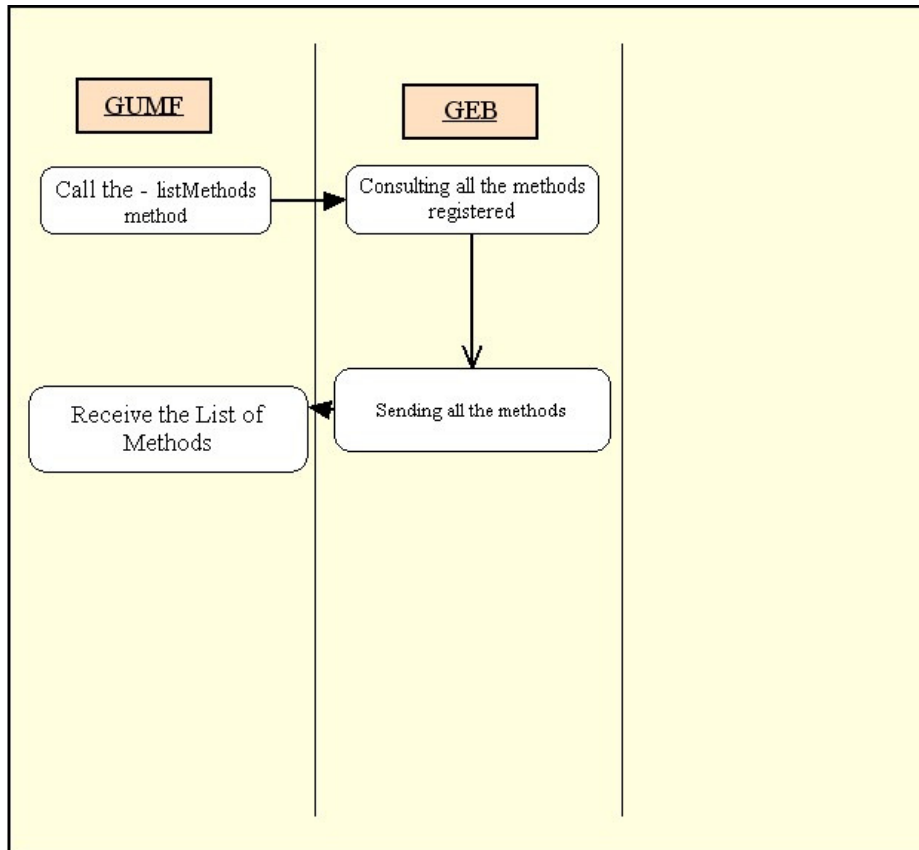


Figure 4: Statechart example of the *listMethods* service.

- *eventGEBListener (previousIdEvent, method, eventBody):String (idAssignedEvent)* (ref to 5.2)

The *eventGEBListener* method is a request-response service. It means that the *eventListener* component sends an event request and it will wait for a response, the *idAssignedEvent*.

This event is a request to the event bus. The event will be processed by GEB, and it will generate another event with additional information like the *idAssignedEvent*.

- *previousIdEvent*, a unique identifier of type String. It is not a mandatory attribute. In this case, this ID is provided by the component which made the request. This ID is the identification, if the request has any relation with other event.
- *method*, the type of event (String). Used by the Event Bus to decide to which listening components an event should be sent. An example of a method to set user model data would be "*setUMData*" (see below to see a more detailed description of the *setUMData* service). The method refers to a specific method that is made available to the system by one of the registered components.
- *body*, contains the actual event call that is interpreted by the components that receive the event. For example, if the component that sends the event wants to set user information in the GUMF, the type of the event would typically be "*setUMData*" (a method provided by the GUMF) and the *body* would contain the parameters for the *setUMData* method, i.e. *userID*, *requestingApplicationID*, *type* and *grappleStatements*. *body* is of type String.

In order to use these Event Bus services, the listening components should implement an *eventListener*. This *eventListener* will be very similar that the Grapple Event Bus has, it will have implemented the functionality *eventEventListener(eventID, previousIdEvent, method, eventBody):void*, but it will be one way service, it means that the GEB will make a request, but it will not wait for a response. The *eventListener* must use the *register* service of the EventBus to subscribe its methods.

- *eventEventListener (eventId, previousIdEvent, method, body):void* (ref to 5.1)

The *eventEventListener* method is a one way service. It means that the *GEB* sends a request to the Listener Components, but it will not wait for a response.

This event is a request to the Listener Components which are registered with the method involved in the *eventEventListener*. This event will be processed by the listening components.

- *eventId*, a unique identifier of events of type String. The *eventId* is generated by GEB. The functionality of this ID, is to provide the possibility to identify the messages exchanged between the components through the GRAPPLE Event Bus,
- *previousIdEvent*, a unique identifier of type String. It is not a mandatory attribute. In this case, this ID is provided by the component which made the request. This ID is the identification, if the request has any relation with other event.
- *method*, the type of event (String). Used by the Event Bus to decide to which listening components an event should be sent. An example of a method to set user model data would be "*setUMData*" (see below to see a more detailed description of the *setUMData* service). The method refers to a specific method that is made available to the system by one of the registered components.
- *body*, contains the actual event call that is interpreted by the components that receive the event. For example, if the component that sends the event wants to set user information in the GUMF, the type of the event would typically be "*setUMData*" (a method provided by the GUMF) and the *body* would contain the parameters for the *setUMData* method, i.e. *userID*, *requestingApplicationID*, *type* and *grappleStatements*. *body* is of type String.

The following sequence diagram abstractly exemplifies the usage of the *eventGEB* method of the Event Bus, and the *eventEventListener* method of the Listener Components. In the example a component called LMS1 wants to query the GUMF for the user data itself stored in the GUMF. Therefore it sends an event to the GEB. It indicates that it wants to use the method *queryUMdata* and it includes the right parameters like the actual query (the GUMF method is explained further on in more detail). The Event Bus computes that only the GUMF and the LMS2 has the method *queryUMData* registered so it forwards the event to the GUMF and the LMS2. GEB generates a response with the id generated for the event. The LMS2 receives and parses the event too, but it considers that the information content in the message is not relevant, and as a result it ignores the event. The GUMF receives and parses the event and generates a new event, including the ID of the previous event, facilitating the identification of the message from the LMS1, and the result for the query in the form of GRAPPLE statements. The GEB computes that this new event from GUMF should be forwarded to LMS1 and LMS2.

In this example, LMS1 call *eventGEBListener(queryUMData, age): 1* where 1 is the *idAssignedEvent*. GUMF call, as response, *eventGEBListener(1,getUMData, 32): 2*.

When LMS1 receives the event, it understands that this response comes from the before event, because it knows that the event LMS1 generated has the *idAssignedEvent* equal to 1.

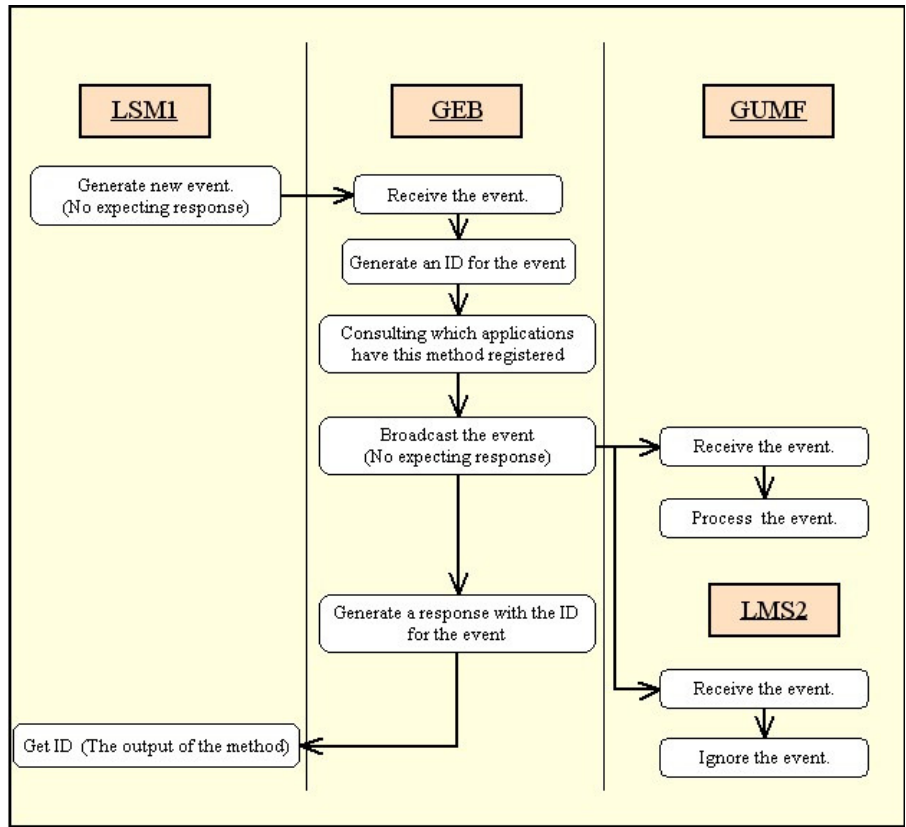


Figure 5: Sequence diagram example of using asynchronous event method over GRAPPLE Event Bus. (First part)

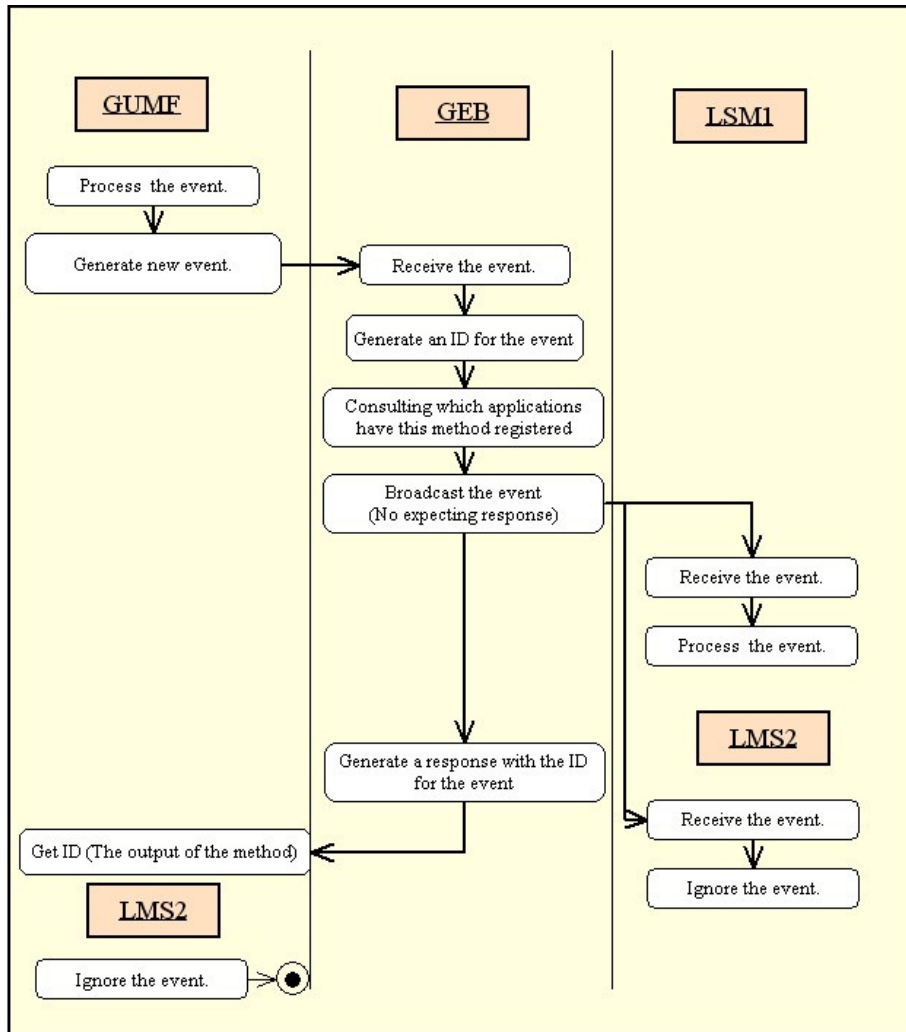


Figure 6: Sequence diagram example of using asynchronous event method over GRAPPLE Event Bus. (Second part)

3.3 GRAPPLE User Model Framework – GUMF (WP2 & WP6)

3.3.1 Component functionality

Working on a centralized GUMF seems to be a step back into the 1990s, during which several UM servers and ontologies 'for all' were designed. However, we will have a different approach where we try to learn from mistakes in the past. The main observation here is that for many reasons a central monolithic UM ontology is neither desirable nor feasible. Therefore, rather than providing a centralized user modeling functionality, we aim at providing a framework for sharing and reasoning about user data, delivered by external (adaptive) systems. Note that the purpose of the GUMF is not, to take over all user modeling facilities of the ALEs and LMSs that will be serviced, but more to be an additional service that allows exchanging UM information. However, if some ALE or LMSs want to use the GRAPPLE UM Framework as its primary UM server this should be possible.

A basic ontology will be provided, to be extended when the need arises. This ontology should model the most common user information for ALEs and LMSs. The purpose of this ontology is to provide a simple system to quickly provide general user information for new connecting system. The basic user model will contain, next to basic interaction data and demographics, elements that may be the result of several lines of reasoning. The reasoning history should be logged, for assessment of the trustworthiness of the inferences - a provenance system. Next to the basic ontology the GUMF can maintain translations that describe how information should be transformed from and to an application specific language (i.e. modeling wise, not

syntax wise). These translations can specify either how incoming information can be translated into the structure that matches the basic ontology and the other way around, or can provide direct translations between a source and target system to provide for the exchange of specific information that is not covered by the basic ontology, but that is shared between two or more applications.

3.3.2 Services offered to other components

The interface of the GRAPPLE UM Framework is implemented by three basic interfaces:

- *tel*: either
 - provide the framework with specific user information
 - inform the framework about an event, to be stored in the core UM
- *ask*: ask for data in the core UM
- *inform*: listener on the framework that informs about events in the framework (like changes for some user in the UM).

By accessing these interfaces, external applications can provide data or make use of available data. Similar to Java, systems may re-implement these interfaces to provide extra functionality - for example some reasoning on usage data. By doing so, the system serves as a plugin that extends the core system.

The GUMF is an example application that registers itself on the GRAPPLE Event Bus and implements a number of methods. GUMF is also part of the core GRAPPLE framework.

For collaboration by exchanging user model data, both the GRAPPLE Core Engine and the LMSs should share their user data with the GUMF. GUMF models user information in GRAPPLE statements, i.e. applications that use the GUMF should also be able to interpret these GRAPPLE statements. GRAPPLE statements are further detailed in [17].

The GUMF implements (and thus only listens to) the following methods:

One time setup actions

- *registerUMApplication (administratorID, administratorSecret, name, url, description)*
 - The one-time registration of a new application to GUMF. On registration, GUMF generates (1) a GUMF ID for the registered application (clientID), (2) a secret which will be shared between GUMF and the client application (sharedSecret), and (3) a default dataspace where the applications can store Grapple statements. The method expects the following parameters as input:
 - *administratorID (String)*: The user ID of the client application administrator, who manages the client application via the GUMF administration interface.
 - *administratorSecret (String)*: A secret/token that allows GUMF to verify that it is really the administrator (identified via administratorID) who wants to register a new client application.
 - *name (String)*: The application name.
 - *url (URL)*: The URL of the client application that should be registered.
 - *description (String)*: A human-readable description of the application. Useful for new application designers to know what applications are subscribed to GUMF.
- *setUMInferenceRules (ID, secret, dataspaceID/pluginID, inferenceRules)*
 - Sets inference rules in the GUMF, which allows the GUMF to infer new data from the data for that application in the database (besides the basic RDFS/OWL inference rules). If these rules are not set the GUMF is not able to infer new knowledge based on the information in the GUMF. Inference rules can either be added to a specific dataspace or to a specific plugin.

- *ID (String)*: Identifies the entity that wants to adjust the inference rules of a dataspace or a particular plugin.
- *secret (String)*: The secret/token of the entity that tries to set the inference rules.
- *dataspaceID/pluginID*: The ID (usually a URI) of the dataspace or plug-in to which the inference rule should be added.
- *inferenceRules (List of String)*: A set of rules that specifies how new data can be inferred. Initial examples of inference rules are given in GRAPPLE Deliverable D2.1 and a specification of inference rules will be given in Deliverable 2.3. Inference rules also allow mapping between different vocabularies, i.e. to transform data into a specific data model.

Operational GUMF Services

- *setUMData (ID, secret, dataspaceID, grappleStatements)*
 - Adds new data (Grapple statements) into a specific dataspace in GUMF. The following parameters are required:
 - *ID (String)*: The ID (as generated by GUMF on registering the client application) of the client application that wants to store Grapple statements.
 - *secret (String)*: The secret that allows (together with the ID) GUMF to authenticate the client application that wants to add data to GUMF.
 - *dataspaceID (String)*: the ID (URI) of the dataspace where the data should be stored (e.g. the ID of the default dataspace that was generated when the client was registered).
 - *grappleStatements (List of Grapple statements)*: The Grapple statements to store. Here, only the user, predicate, object, level and (optionally) origin fields have to be set (cf. Deliverable 2.1 for more details on Grapple statements).
- *queryUMData (ID, secret, dataspaceID, grappleStatements)*
 - Queries for data in the GUMF and requires the following parameters:
 - *ID (String)*: The ID (as generated by GUMF on registering the client application) of the client application that wants to query for Grapple statements.
 - *secret (String)*: The secret that allows (together with the ID) GUMF to authenticate the client application that wants to query for data made available by a specific dataspace in GUMF.
 - *dataspaceID (String)*: the ID (URI) of the dataspace which makes the data available that should be queried (e.g. the ID of the default dataspace that was generated when the client was registered).
 - *query (String)*: Specifies which data should be delivered. Currently, GUMF supports three different query languages, namely SPARQL, SeRQL and a pattern based query language, where the enquirer basically specifies a pattern the returned Grapple statements have to match.

3.3.3 Services required to other components

In order to let systems make use of plugins or to build upon each others' internal user models, systems need to register to the UM Framework. Registration involves the following steps:

- provide a unique name, short description and link to the system's Website
- check prerequisites for proper functioning (i.e. if the system builds upon other systems being registered to the UM framework, these other systems need to be registered)
- provide interface to re-implemented tell, ask and inform functionality, if any.

3.4 GRAPPLE Adaptive Learning Engine – GALE (WP1)

3.4.1 Component Functionality

The GRAPPLE Adaptive Learning Engine (GALE) acts as a server to make adaptive courses available. Java servlets are used to transparently adapt the content to the current user model. Browsing through a course generates new page requests to GALE, which interprets the requests as events that may update the user model (changing things like knowledge, interest, suitability, etc.).

GALE uses plug-ins to adapt various file formats. By default there are plug-ins to adapt xml and more specifically xhtml. There is also a plug-in that converts html to xhtml. Some plug-ins adapt specific tags in an xml document, while others adapt an entire document to, for instance, contain layout (also possibly based on the user model).

GALE can be extended to retrieve domain related information from any source. By default it supports storage in a database (through Hibernate), in old AHA3 files and in GDOM XML files. The domain related information contains all the instructions used by the adaptive engine to reason on the user model and to update the user model. These instructions are expressed in Java code mixed with pseudo code (to easily access domain information and user model).

Though GALE exposes existing courses through servlets, some of GALE's functions can be accessed through SOAP web services.

3.4.2 Services offered to other components

GALE offers web services for adding new courses and retrieving the available courses. For each course GALE can be asked to generate a URL that serves as an entry point to start the course. The course itself is made available through servlets and GALE acts as any normal web application. The access to the resource can be made secure by Shibboleth.

Adding courses

- *boolean addGALModel(String model)*
 - o Creates or updates part of the GALE DM based on the model provided. The model is the serialized XML GAL code and its associated XML serialized DM and UM (bundled in one file). Returns 'true' if the information in the model was successfully added, 'false' otherwise. The details depend on the outcomes from the GAT.

Retrieving courses

- *List<String> getCourses()*
 - o Returns a list of all known courses in IMS LIP serialized XML with learning activity descriptions as defined in D7.2b. Be aware that this method does not scale well, because retrieving the information about ALL courses in GALE might be slow.
- *List<String> getCourses(int first, int count)*
 - o Similar as getCourses. Returns count courses starting with the course indexed by first in the list of courses. Since List<String> getCourses() can cross performance issues, List<String> getCourses(int first, int count) allows to avoid the above performance issues.
- *int getCourseCount()*
 - o Returns the number of registered courses.

3.4.3 Services required to other components

GALE makes use of the GRAPPLE User Modeling Framework (GUMF) to query external user model variables and to make some of its own user model variables public. GALE is mainly interested to have services to store and retrieve GRAPPLE statements.

3.5 GRAPPLE Authoring Tool – GAT (WP3)

In education students generally benefit from personalised attention by their teachers. Often teachers are however unable to provide much, personalised, attention to each student however, due to time constraints, or other constraints. At the same time, on-line courses are becoming increasingly popular and the use of so called Learning Management Systems (LMS) is becoming more widespread. Adaptive Hypermedia (AH) is perceived to have the potential to offer a richer learning experience, personalised for each learner. To realise this potential however, education authors need to have the ability to easily create adaptive material. The GRAPPLE project aims to integrate an AH with major Learning Management Systems and to provide an environment that delivers personalised courses in a LMS interface.

However, designing an AH is a much more complex and time-consuming task, than creating a course in a LMS. There exist several Adaptive Hypermedia reference models, like AHAM (Adaptive Hypermedia Application Model) [13] and LAOS (Layered WWW AH Authoring Model and their corresponding Algebraic Operators) [5] that are specifically developed for authoring (instead of general purpose models of AH, such as XAHM [4], Munich [10], GAHM [11]). However, even when using tools developed based upon these models, authoring remains a difficult and time consuming task. A solution, first proposed in [8], is to use graphical tools. Existing graphical authoring tools (e.g. the Graph Author developed for AHA! [2]) use concrete connections between concepts, and, the adaptivity is specified in a single layer. This approach means that the re-usability of the adaptivity is limited.

To simplify adaptive behaviour authoring for an educational author, a visual environment is considered most intuitive. We have previously introduced the CAM model (Conceptual Adaptation Model) [8], a generic model for authoring that allows adaptation to be defined in a visual, graphical, flexible and user friendly way. In this paper we refine this model and introduce the *design of the new authoring tool*, which allows the specification/creation of a CAM instance in a visual manner. In the overall authoring approach of the GRAPPLE project, tools are being developed to translate these CAM instances into usable adaptive courses for multiple adaptation delivery systems, like, e.g., AHA!. Finally, we introduce a *novel set of adaptation languages* to deal with the novel way of authoring adaptation in a visual fashion.

3.5.1 Component Functionality

3.5.1.1 CAM model

The Concept Adaptation Model (CAM) was introduced for the first time in [8]. In the meantime, after many discussions within the authoring community, some changes were deemed necessary and the model has been refined. In this section we will go into its design in more details. The CAM model is based upon lessons learnt from the AHAM [13] and LAOS [5] models, as well as being inspired by the multi-model, metadata-driven approach to content adaptation [2], and has incorporated ideas from other models, such as Dexter [7], XAHM [4], Munich [10], UWE [9], ADAPT [6].

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

There will always be a DM and UM layer and at least one layer with adaptation aspects, so the structure of the CAM model is a generalization of the AHAM model, and either equivalent with, or a generalization of the more refined LAOS model.

To summarise the CAM model consists of the following layers:

- *Domain Model (DM)*, similar to the AHAM and LAOS DM. The DM in the CAM model only allows domain-specific information (concepts, links and domain-specific meta-data) ,unlike in AHAM;
- *User Model (UM)* similar to the one in AHAM and LAOS.

- a number (at least 1) of *Adaptation Models (AM)*, extending the ideas of generality and specificity: *generic adaptation* rules by means of CRTs: can be applied to specific *concepts in the adaptation* model. Each type of relationship provides a different layer, or view on the course.

Thus, the authoring tool will comprise a DM authoring tool, UM authoring tool, a CRT authoring tool and a CAM authoring tool. The presence of the CRT ensures that the complexity of authoring specification is hidden in the definition of these concept relationships, whilst the presence of the CAM ensures a high-level, non-programming access to authoring behaviour specification. In this paper, due to its important role in integrating all the other authoring elements, as well as due to the lack of space, the focus is on the *CAM model and tool*.

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

3.5.1.2 The Authoring Tool

The Authoring tool is a tool in which authors are able to specify CAM instance models (as such CAM is a meta-modelling system). The author is able to define all layers of their specific CAM model with this tool. The tool consist of three components corresponding to the mandatory layers: DM editing; CRT editing and CAM editing, held together in an integration component. The tool is implemented as a web-based application and was developed in Adobe Flex. The integration component contains general infrastructure for manipulating windows, opening/ saving models and the toolbar, which interacts with each component. Below we see a screenshot of the authoring tool. Below we see a screenshot of the Authoring Tool. The three main components are visible in floating windows, from left to right the DM-tool, CRT-tool and CAM-tool. In the DM-tool we see a graph, with several nodes and links. The nodes represent the concepts in the subject domain and the links represent semantic relationships, without any attached behaviour. Authors can create conceptual Domain Models, by inserting the concepts and their links in the CRT-tool.

The CRT-tool gives the opportunity to edit the adaptive behaviour of a CRT. Authors specify a title and description of the CRT. Many authors with less technical skill will use the CRT-tool mainly to look at the descriptions of and possible User Model variables in use by a CRT. Authors can, if they wish to customise a CRT or create one from scratch, specify the number of sockets; create a list of User Model values that can be used in the code, and write the actual adaptive behaviour in GAL code [12]. The CAM-tool instantiates CRTs with concepts. In the screenshot below we see the nodes, which represent sockets and the links. The links with a diamond label represent the CRTs. The round labels indicate that sockets share concepts. Authors can insert CRTs, from the list of existing CRTs, and copy concepts from a Domain Model, in a DM-tool window to instantiate the CRTs.

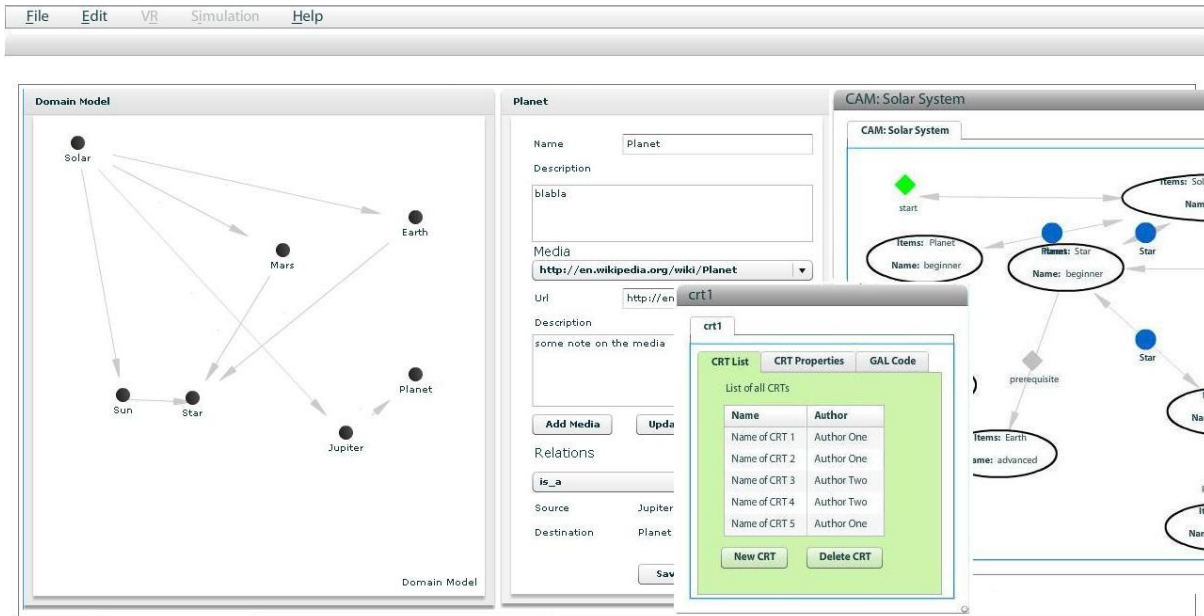


Figure 7: GRAPPLE Authoring Tool.

3.5.2 Services offered to other components

The GRAPPLE Authoring Tool (GAT) offers a web service, *GrappleAuthoringWS*, for storing and retrieving the different models, the DM, CRT and CAM models. In Figure 8 a schematic overview of the web service and its methods is given. Below we will go into detail for each method and give the exact xsd. The whole web service is described in *GrappleAuthoring.wsdl*.

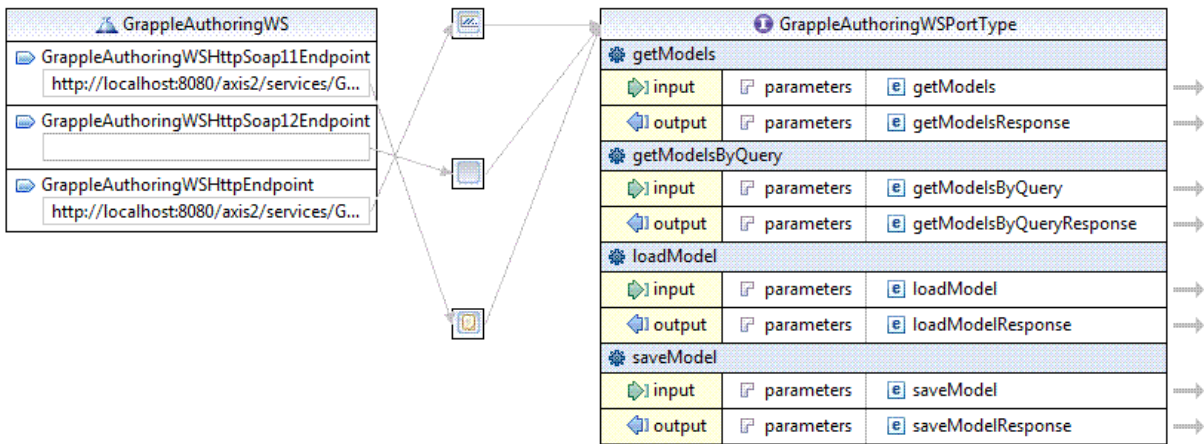


Figure 8: Schematic overview of authoring tool WSDL.

Common header

All the GAT models, the DM, CRT and CAM share a common header. The header contains the following shared information:

- a unique identifier, the *modeluuid*,
- the type of model (*modeltype*) one of *[DM|CRT|VRCRT|CAM|ALL|ALLCRTS]*,
- the id of the author of the model (*authoruuid*)
- the authorisation for other authors (*authorisation*) one of *[nopermission|read|readwrite]*

- the timestamp of the date and time the model was created (*creationtime*)
- the timestamp of the date and time the model was last updated (*updatetime*)
- a title
- a description.

Below we give an example of the header, in this case for a DM:

```
<model>
  <header>
    <modeluuid></modeluuid>
    <modeltype>VRCRT|CRT|DM|CAM</modeltype>
    <authoruuid></authoruuid>
    <!-- authorisation for all other but the author; the author always has all
permissions;
the author defines these flags -->
    <authorisation>[nopermission|read|readwrite]</authorisation>
    <creationtime>unixtimestamp</creationtime>
    <updatetime>unixtimestamp</updatetime>
    <title></title>
    <description></description>
  </header>
  <body>
    <dm>
      <!-- dm information here -->
    </dm>
  </body>
</model>
```

- *getModels (type:String)*

This method gets a list of all available models of a specific type (or all types). It takes 1 string parameter *type*: one of *[DM|CRT|VRCRT|CAM|ALL|ALLCRTS]* and returns an array of headers (see above) in XML format. The xsd is given below:

```
<xs:element name="getModels">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="param0" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="getModelsResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<wsdl:message name="getModelsRequest">
  <wsdl:part name="parameters" element="ns:getModels"/>
</wsdl:message>
<wsdl:message name="getModelsResponse">
  <wsdl:part name="parameters" element="ns:getModelsResponse"/>
</wsdl:message>
```

- *getModelsByQuery (query:String)*

This method returns an array of models in XML format (converted to string) of models that satisfy a given query. It takes a string input parameter, the specific query. At the moment the possible queries have not yet been decided. The method will be used to satisfy specific requests from other components that cannot be solved with the general query mechanisms. The xsd is given below:

```

<xs:element name="getModelsByQuery">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="param0" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="getModelsByQueryResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<wsdl:message name="getModelsByQueryRequest">
  <wsdl:part name="parameters" element="ns:getModelsByQuery"/>
</wsdl:message>
<wsdl:message name="getModelsByQueryResponse">
  <wsdl:part name="parameters" element="ns:getModelsByQueryResponse"/>
</wsdl:message>

```

- *loadModel (uuid:String)*

This method retrieves a model from the store, with a given uuid. It takes a string parameter, the uuid of the model and returns the model in xml format, converted to string. The xsd is given below:

```

<xs:element name="loadModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="param0" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="loadModelResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<wsdl:message name="loadModelRequest">
  <wsdl:part name="parameters" element="ns:loadModel"/>
</wsdl:message>
<wsdl:message name="loadModelResponse">
  <wsdl:part name="parameters" element="ns:loadModelResponse"/>
</wsdl:message>

```

- *saveModel (model:String)*

This method saves a model. It takes a string input parameter, the model in xml format converted to string. The xsd is given below:

```
<xs:element name="saveModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="param0" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="saveModelResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return"
type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<wsdl:message name="saveModelRequest">
  <wsdl:part name="parameters" element="ns:saveModel"/>
</wsdl:message>
<wsdl:message name="saveModelResponse">
  <wsdl:part name="parameters" element="ns:saveModelResponse"/>
</wsdl:message>
```

3.5.3 Services required to other components

The GRAPPLE Authoring Tool required the following services from the following tools:

The **GRAPPLE User Modelling Framework (GUMF)** should provide a service for **retrieving** existing User Model variables. The service should return the names and ranges of variables. The User Model contains overlay variables over the DM or CAM as well as other free variables. Therefore the service has as optional input, a concept or a whole course. If no input is given, only free variables will be returned.

The GAT will provide the author with a simple interface to explore the existing UM variables. Therefore the ranges can be in textual format as the GAT will only display this information to the author.

The **GRAPPLE Adaptive Learning Engine (GALE)** should provide a service for **deploying** courses in the CAM (external) format. The GAT should be able to actively start the deploying process. The service should take as input either the entire model as xml (converted to string), or alternatively the uuid of the model, in the repository.

3.6 GRAPPLE Visualisation – GVIS (WP4)

The aim of the GVIS module is to offer some facilities to allow representation of user-related data in the learning process. This visualization will be adapted to the user's role and the user's current state, in order to support a better understanding of the current situation.

3.6.1 Services offered to other components

The module does not offer any Web Service interface to other components, except the eventEventListener.

3.6.2 Services required to other components

The module relies on different components inside the GRAPPLE infrastructure:

- GALE: contains all the data needed to deliver the course and performs adaptation. Some needed data, such as the list of concepts of the course is stored in this module, hence we need to access data provided/handled by GALE
- GUMF: contains all user-related data. It is the main source of data for GVIS.
- CAM: contains the didactical relationship between concepts and conditions for adaptation (such as the expected level of knowledge for concepts) that are defined by the authors.

All of these modules are expected to offer a WS interface, to allow the retrieve of information from other components, like GVIS.

3.7 LMS

3.7.1 Component Functionality

The key success factor of the GRAPPLE project is the interoperability between the adaptive facilities developed by the ALE components and LMSs already existing in academic and industrial settings.

The GRAPPLE consortium has included five different LMSs, three open-source Claroline, Moodle, Sakai and two commercial learn eXact and IMC CLIX.

The LMS is the daily tool used by thousands of learners to access the courses they have been enrolled. The added value of the integration with GRAPPLE system is to allow the learner to take advantage of an innovative system by using their standard LMS.

The existing LMSs are also precious source of user information for the GRAPPLE system, in particular to the GUMF. D7.2b is in charge to describe the interoperability data model of the GRAPPLE system and the specific parameters made available by the LMSs to GUMF.

The five LMSs involved in the GRAPPLE consortium are very different:

- Claroline and Moodle are PHP based
- Sakai and IMC CLIX are Java based
- learn eXact is .NET based.

The GUMF receives the User Model information provided at the following events, as documented in D7.2b [18]:

- first User Login with LMS integrated with GRAPPLE
- student enrolment
- role change
- resource addition/change
- access to a course/learning activity/resource
- test/quiz completion
- user login.

Any event provides a different set of User Model information.

Sakai can provide this data in real time when the above mentioned events happen. The other LMSs need to set up a system that periodically provides the user details update to the GUMF.

3.7.2 Services offered to other components

The LMSs do not offer any Web Service interface to other components, except the *eventEventListener*.

3.7.3 Services required to other components

The LMSs rely on the following GRAPPLE components:

- GUMF: contains all user-related data and it has to provide the web services to access the Grapple statements (described in D7.2a) and to memorise the Grapple statement.
- GALE: has to provide services for **accessing** and **delivering** (through Shibboleth) the adaptive courses.

4 ANNEX A – Shibboleth

4.1 Installation of Shibboleth IdP

For the GRAPPLE project the Shibboleth Identity Provider, version 2.0 has been used. It is a standard Java web application based on the Servlet 2.4 specification. The installation procedure consisted of the following main steps.

1. Download the IdP software package from Internet2 Shibboleth Download site (<http://shibboleth.internet2.edu/downloads/shibboleth/idp/>) and unzip it.
2. Prepare our Apache Tomcat servlet container.
 - Placed .jar files included in the IdP source *endorsed* directory into Tomcat's `$CATALINA_HOME/common/endorsed` directory.
 - Added the parameters to the JAVA_OPTS environment variable to extend the allocated memory (`-Xmx512m, -XX:MaxPermSize=512m`).
 - Copied the connector definition specified below into Tomcat's `conf/server.xml` and placed it after the last connector already defined in the configuration file (where IDP_HOME is the name of directory we installed IdP - `/usr/shibboleth/shibboleth-idp`).

```
<Connector port="8443"
  maxHttpHeaderSize="8192"
  maxSpareThreads="75"
  scheme="https"
  secure="true"
  clientAuth="want"
  SSLEnabled="true"
  sslProtocol="TLS"
  keystoreFile="IDP_HOME/credentials/idp.jks"
  keystorePass="PASSWORD"
  truststoreFile="IDP_HOME/credentials/idp.jks"
  truststorePass="PASSWORD"
  truststoreAlgorithm="DelegateToApplication"/>
```

3. To install IdP, run the script `./install.sh` and specified the location of the IdP home directory during the installation procedure.
4. Define a metadata source for the metadata information of GALE.
5. Deploy the IdP WAR file, located in `IDP_HOME/war/`, after you have properly prepared your servlet container.

After the installation script has been completed the IdP home directory is created.

To perform a first quick test, access the URL: `http://dyn188.win.tue.nl/idp/profile/Status`. The "OK" message confirms that everything has worked correctly.

4.2 Configuration of IdP

The Shibboleth 2.0 IdP uses the following configuration files to control various aspects of its functionality:

- *attribute-filter.xml* – to configure the release of attributes to SP's,
- *attribute-resolver.xml* – to configure attribute collection, transformation, and encoding,
- *handler.xml* - to configure how the IdP receives and responds to various message types,
- *relying-party.xml* – to configure how the IdP processes messages that are received,
- *logging.xml* – to configure of the IdP's logging system,
- *login.config* – to configure the [Username/Password authentication mechanism](#),
- *service.xml* – to configure the coarse grained IdP components,
- *internal.xml* – this is a low-level IdP configuration file.

For the purpose of the GRAPPLE Project the following configuration of Shibboleth IdP has been selected:

2) Configuration for communicating with a Relying Party (RP) (new service provider) - GALE

The configuration for communicating with a new service provider (or more generically, Relying Party (RP)) is located in *\$IDP_HOME/conf/relying-party.xml*. The IdP recognizes three types of RP:

- (1) *anonymous* (<AnonymousRelyingParty>) – for which the IdP has no metadata,
- (2) *default* (<DefaultRelyingParty>) – for which the IdP does have metadata but for which there is no specific configuration, and
- (3) a *specified* RP (<RelyingParty>) – for which the IdP has metadata and a specific configuration.

We determine GALE RP using the *specified* relying party configuration.

When creating a specified RP configuration the following attributes were required:

- **id** - URI of the entity or group to which this configuration applies,
- **provider** - URI of the IdP when answering requests to this RP or group.

The RP also contains a list of supported communication profiles. Each profile configuration is identified by a <ProfileConfiguration> element.

The configuration of GALE Relying Party is the following:

- `<RelyingParty id="http:// dyn188.win.tue.nl:8080/gale" provider="https://dyn188.win.tue.nl/idp/shibboleth">`
- `<ProfileConfiguration xsi:type="saml:ShibbolethSSOProfile" />`
- `<ProfileConfiguration xsi:type="saml:SAML1AttributeQueryProfile" />`
- `<ProfileConfiguration xsi:type="saml:SAML1ArtifactResolutionProfile" />`
 - `<ProfileConfiguration xsi:type="saml:SAML2SSOProfile" signAssertions="never" signResponses="never" encryptAssertions="never" />`
- `<ProfileConfiguration xsi:type="saml:SAML2AttributeQueryProfile" />`
- `<ProfileConfiguration xsi:type="saml:SAML2ArtifactResolutionProfile" />`
- `</RelyingParty>`

3) Define a new Metadata Source and Metadata Provider

SAML metadata information plays an important role in functionality of the Shibboleth IdP. Metadata is provided to the IdP through Metadata Providers which are defined in the *\$IDP_HOME/config/relying-party.xml* file.

The IdP may have one, logical, Metadata Provider per RP group (multiple, individual, Metadata Providers can be chained together to create the logical Provider used). Metadata Providers are defined with a *MetadataProvider* element appearing after all the *RelyingParty* elements in the *relying-party.xml* file. The

Metadata Provider is identified by the `xsi:type` attribute and must have an `id` attribute which assigns a unique id to the provider. The following types of metadata providers are supported in Shibboleth:

(1) Chaining Metadata Provider - allows combining other Metadata Provider definitions; when the IdP requests metadata from this type of provider it returns the response of the first metadata provider, in the chain, that provides a response;

(2) Filesystem Metadata Provider - reads SAML 2 metadata from a file on the file system; metadata is cached in memory for a period of time; the changes in the file are being monitored so the file would be reloaded upon detecting an update;

(3) File Backed HTTP Metadata Provider – works like the HTTP Metadata Provider, but stores metadata in a local temporary file upon successful retrieval to protect against the case where metadata has been successfully fetched once but the remote server is offline during subsequent requests.

(4) HTTP Metadata Provider - reads metadata from an http or https scheme URL. Metadata is cached in memory for a period of time in order to improve performance.

(5) Inline Metadata Provider - reads metadata from with the provider's configuration element.

To define Gale Metadata Provider we used File Backed HTTP type:

- `<MetadataProvider id="GaleMD" xsi:type="FileBackedHTTPMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata" metadataURL="http://dyn188.win.tue.nl:8080/gale/metadata.xml" backingFile="/usr/shibboleth/shibboleth-idp/metadata/gale-metadata.xml" />`

4) Define and Release new attributes in the IdP

An attribute passes through four main steps from source system to release to an SP: (1) pulling from the system of record, (2) transformation and preparation, (3) encoding, and (4) filtering and release. Every attribute has a unique attribute ID which is used to refer to it consistently through this process.

Shibboleth does not store any attributes about users itself; it relies on external data stores to supply user information to be released. These attributes are pulled from data sources using data connectors. Data connectors may produce more than one attribute and each attribute may have many values. Attributes produced by data connectors are never released outside of the attribute resolver. The following types of connectors are supported:

- *Static data connector* is used to add statically attributes and values to every person served by the identity provider (e.g. adding an entitlement attribute for each user).
- *Computed ID data connector* is used to construct unique identifiers by hashing together some information.
- *Stored ID data connector* is used to construct and persist identifiers by means of a database.
- *Relational database connector* is used to pull attributes from a relational database by executing some configured SQL.
- *LDAP data connector* is used to pull attributes from an LDAP directory by executing an LDAP filter on a specific branch.

In our settings we are using Computed ID data Connector.

Each attribute undergoes two rounds of transformation while it passes through the IdP:

- (1) IdP transforms attributes (merge, split, reformat, etc.) using other attributes to get a complete data definition;
- (2) IdP transforms the attribute from the internal representation to one appropriate for the protocol the IdP will be communicating with (attribute encoding).

An attribute definition should be specified to give an ID and value to the attribute. If additional transformation is needed, additional attribute definitions dependent on the primary attribute definition can be used. Once the attribute has the internal name and value, it is being encoded to give it the right format on the wire. Attribute encoders should be attached to the attribute definitions. The defined attributes are not released to SP until an attribute filter policy for that attribute is defined. Such policies describe which service providers, under which conditions, receive which attributes.

An example of the attributed definition used in GRAPPLE IdP is provided below.

- `<resolver:AttributeDefinition xsi:type="ad:Simple" id="uid" sourceAttributeID="NETID">`
- `<resolver:AttributeEncoder xsi:type="enc:SAML2String"`
- `name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"`
- `friendlyName="eduPersonPrincipalName" />`
- `<resolver:Dependency ref="MyDatabase" />`
- `</resolver:AttributeDefinition>`
- `<resolver:DataConnector xsi:type="RelationalDatabase"`
- `xmlns="urn:mace:shibboleth:2.0:resolver:dc"`
- `id="MyDatabase"`
- `validationQuery="SELECT 1; ">`
- `<ApplicationManagedConnection jdbcDriver="org.hsqldb.jdbcDriver"`
- `jdbcURL="jdbc:hsqldb:res:/data/database/shibdb"`
- `jdbcUserName="sa" />`
- `<QueryTemplate>`
- `<![CDATA[`
- `SELECT * FROM PEOPLE WHERE netid='${principal}'`
- `]]>`
- `</QueryTemplate>`
- `</resolver:DataConnector>`

4.2.1 Definition of the authentication services

In order to support SAML 2 authentication request features like passive and forced authentication and authentication contexts the IdP must do some amount of processing prior to invoking whatever mechanism(s) is in place to authenticate users.

Each authentication mechanism is configured differently, as each has different requirements. The following methods are supported:

- (1) Remote User - sets the principal name in the IdP to REMOTE_USER as determined by the web server or container's authentication;
- (2) GRAPPLE Identifier (GID)/Password - presents the user with an authentication page and then checks the entered username and password against an LDAP directory or Kerberos 5 domain;
- (3) IP Address - checks the user's IP address against a given range to identify the user as a pre-determined principal;
- (4) Previous Session - called when the user's existing session is used as the authentication mechanism (i.e. when the IdP is performing an SSO based sign-on.)

In GRAPPLE IdP GID/Password identification mechanism is used.

The IdP supports the use of multiple different authentication methods at one time. In SAML 2.0 a service provider may require one (from a possible list) authentication method be used. If the IdP is configured for one of the methods required by the SP, then such a method is used. If the Service Provider does not require

a specific method, then the default method, identified on the relying party's configuration, is used. If no default is specified, the IdP chooses a method randomly.

If the user has an existing session, that session indicates that there is an active authentication method that meets relying party's requirements (if any), and the PreviousSession login handler is configured than this handler is used. This is what gives Shibboleth its SSO support.

It should be noted that if the active authentication method does not meet the Relying Party's (RP) requirements, then the user will be prompted to authenticate again using the method required by the second RP. This can happen unintentionally if you have one RP that has a specific relying party configuration specifying GID/Password while a second RP uses the default relying party configuration which does not specify a method and ends up using the "unspecified" method handler. The access sequence of RP1, then RP2 will work since RP2 doesn't care what authentication method was used. However, the sequence from RP2 to RP1 will require re-authentication since RP1 wants GID/Password and "unspecified" was used before.

To be a bit more specific here, in this case, Shibboleth will pass in the authentication method as "unspecified". If the login handler in place does not change that value, then Shibboleth will record that method in the session. If the login handler in use updates the method to reflect a different method, then that method will be recorded in the session. So, the overall behavior is dependent not only on the Shibboleth configuration used, but on the login handler used.

5 ANNEX B - Service Interface Guidelines & GRAPPLE WSDL files

All WSDL [2] interface specifications used in GRAPPLE need to share a common coding style ensuring syntactical homogeneity between the different modules. These coding styles include guidelines for general formatting and typesetting, naming conventions and structural guidelines for exchanged SOAP messages.

By convention, all names in GRAPPLE WSDLs use CamelCase.

Notation: Names of ports, services, bindings and non-anonymous schema types start with capital letters, all other names with lower case letters.

WSDL files consist of 5 primary structural sections:

- Type definitions containing XML Schema definitions (xsd) [1]
- Service definitions containing a reference to the service which is to be exposed
- Port definitions defining ports and their operations. Also, the messages consumed and produced by these operations are stated.
- Binding definitions linking services to ports and providing additional information about the implementation and encoding of the operations.
- Message definitions of those messages used by operations.

The Web Services are described by two WSDL documents:

- NameWSDL.wsdl: It will describe the messages referencing its schemas, the ports and its operations.
- NameWSDLWrapper.wsdl: This WSDL document imports the NameWSDL.wsdl document using the import element from WSDL schema. This document contents the definition of the bindings which doing references to the messages and the ports.

This section provides the WSDL files for GRAPPLE to define the webservices and related operations able to support the integration of the single GRAPPLE components with the GRAPPLE Event Bus.

5.1 eventEventListener

eventEventListener is a web service located on the *eventListener* present at the GAT, GUMF, LMS and GVIS sides. It processes the events that have been rerouted by the GRAPPLE Event Bus to all the GRAPPLE components.

5.1.1 eventDispatchedRequestMsg.xsd

The following xsd describes the request message managed by the *eventEventListener* webservice.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema xmlns:tns="http://xml.netbeans.org/schema/entListenerDefinition"
elementFormDefault="qualified"
targetNamespace="http://xml.netbeans.org/schema/entListenerDefinition"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="eventRequestMsg">
    <xsd:sequence>
      <xsd:element name="eventId" type="xsd:string" />
      <xsd:element minOccurs="0" name="previousIdEvent" type="xsd:string" />
      <xsd:element name="method" type="xsd:string" />
      <xsd:element name="body" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="eventRequestMsg" type="tns:eventRequestMsg" />
</xsd:schema>
```

5.1.2 eventEventListener.wsdl

The following wsdl describes the *eventEventListener* messages referencing its schemas, the ports and its operations.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="eventEventListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener"
  xmlns:ns="http://xml.netbeans.org/schema/entListenerDefinition"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <types>
    <xsd:schema
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener">
      <xsd:import
namespace="http://xml.netbeans.org/schema/entListenerDefinition"
schemaLocation="eventDispatchedRequestMsg.xsd"/>
    </xsd:schema>
  </types>
  <message name="eventEventListenerOperationRequest">
    <part name="eventRequestMsg" type="ns:eventRequestMsg"/>
  </message>
  <portType name="eventEventListenerPortType">
    <operation name="eventEventListenerOperation">
      <input name="input1"
message="tns:eventEventListenerOperationRequest"/>
    </operation>
  </portType>
  <plnk:partnerLinkType name="eventEventListener">
    <!-- A partner link type is automatically generated when a new port type
is added. Partner link types are used by BPEL processes.
```

In a BPEL process, a partner link represents the interaction between the BPEL process and a partner service. Each partner link is associated with a partner link type.

A partner link type characterizes the conversational relationship between two services. The partner link type can have one or two roles.-->

```
<plnk:role name="eventEventListenerPortTypeRole"
portType="tns:eventEventListenerPortType"/>
</plnk:partnerLinkType>
</definitions>
```

5.1.3 eventEventListenerWrapper.wsdl

The following wsdl file imports the previous wsdl document using the import element from WSDL schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener"
xmlns:ns="http://xml.netbeans.org/schema/entListenerDefinition"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
name="eventEventListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener">
<wsdl:import namespace="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener"
location="eventEventListener.wsdl"/>
<wsdl:binding name="eventEventListenerBinding"
type="tns:eventEventListenerPortType">
<soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
<wsdl:operation name="eventEventListenerOperation">
<soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="eventEventListenerOperation_action"/>
<wsdl:input name="input1">
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/eventEventListener"
use="literal"/>
</wsdl:input>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="eventEventListenerService">
<wsdl:port name="eventEventListenerPort"
binding="tns:eventEventListenerBinding">
<soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="REPLACE_WITH_ACTUAL_URL"/>
</wsdl:port>
</wsdl:service>
</definitions>
```

5.2 eventGEBListener

eventGEBListener is a web service provided by the GRAPPLE Event Bus. It generates an event with additional information to be rerouted by the GRAPPLE Event Bus to all the GRAPPLE components.

5.2.1 eventRequestMsg.xsd

The following xsd describes the request message managed by the *eventGEBListener* webservice.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema xmlns:tns="http://xml.netbeans.org/schema/entListenerDefinition"
elementFormDefault="qualified"
```

```
targetNamespace="http://xml.netbeans.org/schema/entListenerDefinition"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="eventRequestMsg">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="previousIdEvent" type="xsd:string" />
      <xsd:element name="method" type="xsd:string" />
      <xsd:element name="body" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="eventRequestMsg" type="tns:eventRequestMsg" />
</xsd:schema>
```

5.2.2 eventResponseMsg.xsd

The following xsd describes the response message provided by the *eventGEBListener* webservice.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema xmlns:tns="http://xml.netbeans.org/schema/eventResponseMsg"
elementFormDefault="qualified"
targetNamespace="http://xml.netbeans.org/schema/eventResponseMsg"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="eventResponseMsg">
    <xsd:sequence>
      <xsd:element name="idAssignedEvent" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="eventResponseMsg" type="tns:eventResponseMsg" />
</xsd:schema>
```

5.2.3 eventGEBListener.wsdl

The following wsdl describes the *eventGEBListener* messages referencing its schemas, the ports and its operations.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="eventGEBListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
  xmlns:ns0="http://xml.netbeans.org/schema/eventResponseMsg"
  xmlns:ns="http://xml.netbeans.org/schema/entListenerDefinition"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <types>
    <xsd:schema
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener">
      <xsd:import
namespace="http://xml.netbeans.org/schema/entListenerDefinition"
schemaLocation="eventRequestMsg.xsd"/>
      <xsd:import
namespace="http://xml.netbeans.org/schema/eventResponseMsg"
schemaLocation="eventResponseMsg.xsd"/>
    </xsd:schema>
  </types>
  <message name="eventGEBListenerOperationRequest">
    <part name="part1" type="ns:eventRequestMsg"/>
  </message>
  <message name="eventGEBListenerOperationResponse">
    <part name="part1" type="ns0:eventResponseMsg"/>
  </message>
  <portType name="eventGEBListenerPortType">
```

```

    <operation name="eventGEBListenerOperation">
      <input name="input1"
message="tns:eventGEBListenerOperationRequest"/>
      <output name="output1"
message="tns:eventGEBListenerOperationResponse"/>
    </operation>
  </portType>
  <plnk:partnerLinkType name="eventGEBListener">
    <!-- A partner link type is automatically generated when a new port type
is added. Partner link types are used by BPEL processes.
In a BPEL process, a partner link represents the interaction between the BPEL
process and a partner service. Each partner link is associated with a partner
link type.
A partner link type characterizes the conversational relationship between two
services. The partner link type can have one or two roles.-->
    <plnk:role name="eventGEBListenerPortTypeRole"
portType="tns:eventGEBListenerPortType"/>
  </plnk:partnerLinkType>
</definitions>

```

5.2.4 eventGEBListenerWrapper.wsdl

The following wsdl file imports the previous wsdl document using the import element from WSDL schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
xmlns:ns0="http://xml.netbeans.org/schema/eventResponseMsg"
xmlns:ns="http://xml.netbeans.org/schema/entListenerDefinition"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
name="eventGEBListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener">
  <wsdl:import namespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
location="eventGEBListener.wsdl"/>
  <wsdl:binding name="eventGEBListenerBinding"
type="tns:eventGEBListenerPortType">
    <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <wsdl:operation name="eventGEBListenerOperation">
      <soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="eventGEBListenerOperation_action"/>
      <wsdl:input name="input1">
        <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
use="literal"/>
      </wsdl:input>
      <wsdl:output name="output1">
        <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/eventGEBListener"
use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="eventGEBListenerService">
    <wsdl:port name="eventGEBListenerPort"
binding="tns:eventGEBListenerBinding">
      <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="REPLACE_WITH_ACTUAL_URL"/>
    </wsdl:port>
  </wsdl:service>
</definitions>

```

5.3 GebListener

GebListener is a web service provided by the GRAPPLE Event Bus to manage the registration and the list of methods that can required to one of the GRAPPLE components (GALE, GAT, GUMF or GVIS).

5.3.1 registerEventListenerRequestMsg.xsd

The following xsd describes the request message managed by the *gebListener* webservice for the registration.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema
xmlns:tns="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
elementFormDefault="qualified"
targetNamespace="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="registerEventListenerRequestMsg">
    <xsd:sequence>
      <xsd:element name="eventListenerID" type="xsd:string" />
      <xsd:element maxOccurs="unbounded" name="methods" type="tns:methods" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="methods">
    <xsd:sequence>
      <xsd:element name="method" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="registerEventListenerRequestMsg"
type="tns:registerEventListenerRequestMsg" />
</xsd:schema>
```

5.3.2 registerEventListenerResponseMsg.xsd

The following xsd describes the response message provided by the *gebListener* webservice for the registration.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema
xmlns:tns="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
elementFormDefault="qualified"
targetNamespace="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="registerEventListenerResponseMsg">
    <xsd:sequence>
      <xsd:element name="responseMsg" type="xsd:boolean" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="registerEventListenerResponseMsg"
type="tns:registerEventListenerResponseMsg" />
</xsd:schema>
```

5.3.3 listMethodsResponseMsg.xsd

The following xsd describes the message provided by the *gebListener* webservice with the list of methods registered from all the component eventListeners.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```

<!--Created with Liquid XML Studio 6.1.18.0 - FREE Community Edition
(http://www.liquid-technologies.com)-->
<xsd:schema xmlns:tns="http://xml.netbeans.org/schema/listMethodsResponseMsg"
elementFormDefault="qualified"
targetNamespace="http://xml.netbeans.org/schema/listMethodsResponseMsg"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="methodsInformation">
    <xsd:sequence>
      <xsd:element name="eventListenerID" type="xsd:string" />
      <xsd:element name="methods" type="tns:methods" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="methods">
    <xsd:sequence>
      <xsd:element name="method" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

5.3.4 gebListener.wsdl

The following wsdl describes the *gebListener* messages referencing its schemas, the ports and its operations.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="gebListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
  xmlns:ns1="http://xml.netbeans.org/schema/listMethodsResponseMsg"
  xmlns:ns0="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
  xmlns:ns="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <types>
    <xsd:schema
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener">
      <xsd:import
namespace="http://xml.netbeans.org/schema/listMethodsResponseMsg"
schemaLocation="listMethodsResponseMsg.xsd"/>
      <xsd:import
namespace="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
schemaLocation="registerEventListenerResponseMsg.xsd"/>
      <xsd:import
namespace="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
schemaLocation="registerEventListenerRequestMsg.xsd"/>
    </xsd:schema>
  </types>
  <message name="gebListenerOperationRequest">
    <part name="registerEventListenerRequestMsg"
type="ns:registerEventListenerRequestMsg"/>
  </message>
  <message name="gebListenerOperationResponse">
    <part name="registerEventListenerResponse"
type="ns0:registerEventListenerResponseMsg"/>
  </message>
  <message name="gebListenerOperationRequest1"/>
  <message name="gebListenerOperationResponse1">
    <part name="listMethodsResponse" type="ns1:methodsInformation"/>
  </message>
  <portType name="gebListenerPortType">

```

```

<operation name="gebRegisterListenerOperation">
  <input name="input1" message="tns:gebListenerOperationRequest"/>
  <output name="output1" message="tns:gebListenerOperationResponse"/>
</operation>
<operation name="gebListMethodsListenerOperation">
  <input name="input2" message="tns:gebListenerOperationRequest1"/>
  <output name="output2" message="tns:gebListenerOperationResponse1"/>
</operation>
</portType>
<plnk:partnerLinkType name="gebListener">
  <!-- A partner link type is automatically generated when a new port type
is added. Partner link types are used by BPEL processes.
In a BPEL process, a partner link represents the interaction between the BPEL
process and a partner service. Each partner link is associated with a partner
link type.
A partner link type characterizes the conversational relationship between two
services. The partner link type can have one or two roles.-->
  <plnk:role name="gebListenerPortTypeRole"
portType="tns:gebListenerPortType"/>
</plnk:partnerLinkType>
</definitions>

```

5.3.5 gebListenerWrapper.wsdl

The following wsdl file imports the previous wsdl document using the import element from WSDL schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
xmlns:ns1="http://xml.netbeans.org/schema/listMethodsResponseMsg"
xmlns:ns0="http://xml.netbeans.org/schema/registerEventListenerResponseMsg"
xmlns:ns="http://xml.netbeans.org/schema/registerEventListenerRequestMsg"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" name="gebListener"
targetNamespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener">
<wsdl:import namespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener"
location="gebListener.wsdl"/>
<wsdl:binding name="gebListenerBinding" type="tns:gebListenerPortType">
<soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <wsdl:operation name="gebRegisterListenerOperation">
    <soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="gebRegisterListenerOperation_action"/>
    <wsdl:input name="input1">
      <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener" use="literal"/>
    </wsdl:input>
    <wsdl:output name="output1">
      <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener" use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="gebListMethodsListenerOperation">
    <soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="gebListMethodsListenerOperation_action"/>
    <wsdl:input name="input2">
      <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener" use="literal"/>
    </wsdl:input>
    <wsdl:output name="output2">
      <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
namespace="http://j2ee.netbeans.org/wsdl/GEB/gebListener" use="literal"/>
    </wsdl:output>
  </wsdl:operation>

```

```

        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="gebListenerService">
    <wsdl:port name="gebListenerPort" binding="tns:gebListenerBinding">
        <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="REPLACE_WITH_ACTUAL_URL"/>
    </wsdl:port>
</wsdl:service>
</definitions>

```

References

1. W3C XML Schema (XSD), <http://www.w3.org/XML/Schema>
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1, March 15, 2001, <http://www.w3.org/TR/wsdl>
3. Brooke, J.: SUS: a "quick and dirty" usability scale. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A. and McClelland, A.L. (eds.). Usability Evaluation in Industry. London: Taylor and Francis. 1996
4. Cannataro, M., Pugliese, A.: XAHM: an XML-based Adaptive Hypermedia Model and its Implementation, 3rd Workshop on Adaptive Hypertext and Hypermedia (AH2001) in conjunction with Twelfth ACM Conference on Hypertext and Hypermedia, Aarhus, Denmark (2001)
5. Cristea, A.I., de Mooij, A.: LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators, WWW'03, The Twelfth International World Wide Web Conference, Alternate Track on Education, Budapest, Hungary (2003)
6. Garzotto F. and Cristea A.I.: ADAPT: Major design dimensions for educational adaptive hypermedia. ED-MEDIA 2004 Conference, AACE, Lugano, Switzerland, June (2004)
7. Halasz F., Schwartz, M.: The Dexter hypertext reference model: Hypermedia. Communications of the ACM, 37(2), 30-39 (1994)
8. Hendrix, M., De Bra, P., Pechenizkiy, M., Smits, D., Cristea, A. I.: Defining adaptation in a generic multi layer model: CAM: The GRAPPLE Conceptual Adaptation Model, In Converging from different backgrounds proceedings of Third European Conference on Technology Enhanced Learning ECTEL (2008)
9. Koch N., Wirsing, W.: Software Engineering for Adaptive Hypermedia Applications? 3rd Workshop on Adaptive Hypertext and Hypermedia, UM 2004 Conference (2001)
10. Koch N., Wirsing M.: Software Engineering for Adaptive Hypermedia Applications. 8th International Conference on User Modeling, Sonthofen, Germany (2001)
11. Ohene-Djan J. A Formal Approach to Personalisable, Adaptive Hyperlink-Based Interaction. PhD thesis, Department of Computing, Goldsmiths College, University of London (2000)
12. van der Sluijs, K., Hidders, J., Leonardi, E., Houben G.J.: Generic Adaptation Language for describing Adaptive Hypermedia, Proc of International Workshop on Dynamic and Adaptive Hypertext: Generic Frameworks, Approaches and Techniques (2009)
13. Wu, H.: A Reference Architecture for Adaptive Hypermedia Applications, doctoral thesis, Eindhoven University of Technology, The Netherlands, ISBN 90-386-0572-2 (2002)
14. Documentation of Shibboleth from Shibboleth Official Web Site. <http://shibboleth.internet2.edu>
15. Materials of Shibboleth Install Fest. Available Online at <http://www.switch.ch/aai/support/presentations/installfest-2009/resources.html>
16. Abel F., Herder F.: D2.1 – Definition of an appropriate User Profile format, <http://www.grapple-project.org/workspace/work-packages-workspace/wp2-user-modeling-methods-and-techniques/t2.1-definition-of-an-ontology-based-user-model/deliverable-drafts/Grapple-WP2-D2-1-User-profile-format.doc/view>
17. Oneto L., Heckmann D.: D7.2a – Data models and related documentation – first version, <http://www.grapple-project.org/workspace/work-packages-workspace/interfacing-between-ale-and-lms/T7.2/d7.2.a-data-models-and-related-documentation-first-version/D7-2a-v1.0.doc/view>
18. Oneto L, et al: D7.2b – Data models and related documentation – update.