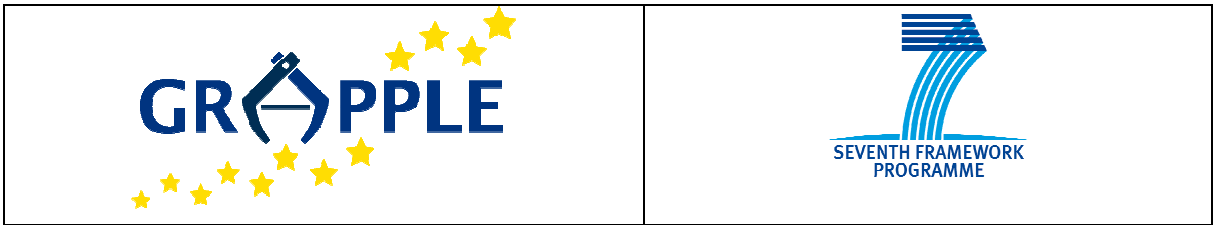

# GRAPPLE

D1.2a Version: 1.0

## Resource Selector Specification

| | |
|---|---|
| **Document Type** | Deliverable |
| **Editor(s):** | Kees van der Sluijs (TUE) |
| **Author(s):** | Paul De Bra (TUE), David Smits (TUE), Kees van der Sluijs (TUE), Evgeny Knutov (TUE) |
| **Internal Reviews** | Andrea Lorenzon (GILABS), Dominik Heckmann (DFKI) |
| **Work Package:** | 1 |
| **Due Date:** | 01-02-2009 |
| **Version:** | 1.0 |
| **Version Date:** | 25-02-2009 |
| **Total number of pages:** | 9 |

**Abstract:** This deliverable describes the "translation" from concepts to resources in GRAPPLE. Access to information, activities, tasks, etc. is done through *concepts*. Each concept may be linked with a number of concrete resources (files, pages, queries to repositories or anything else that can be addressed through a URI). Based on user model properties a runtime selection of a resource or query is made.

**Keyword list:** Resource Selector, GAL, GALE

# Summary

This deliverable describes the "translation" from concepts to resources in GRAPPLE. Access to information, activities, tasks, etc. is done through *concepts*. Each concept may be linked with a number of concrete resources (files, pages, queries to repositories or anything else that can be addressed through a URI). Based on user model properties a runtime selection of a resource or query is made. This is what the AHAM reference model [1] calls the *page selector*. In particular we show how this page selection process is executed by the GALE engine (described in deliverable D1.3a) and how it can be specified in the GAL language (described in deliverable D1.1a).

# Authors



| Person | Email | Partner code |
|---|---|---|
| **Paul De Bra** | debra@win.tue.nl | TUE |
| **David Smits** | d.smits@tue.nl | TUE |
| **Evgeny Knutov** | e.knutov@tue.nl | TUE |
| **Kees van der Sluijs** | k.a.m.sluijs@tue.nl | TUE |

# Table of Contents



# Tables and Figures

**List of Figures**



# List of Acronyms and Abbreviations

| | |
|---|---|
| AHA! (or AHA) | Adaptive Hypermedia Architecture (also used as prefix for other terms) |
| ALE | Adaptive Learning Environment |
| CAM | Conceptual Adaptation Model |
| AM | Adaptation Model |
| DM | Domain Model (this includes the Adaptation Model) |
| GRAPPLE | Generic Responsive Adaptive Personalized Learning Environment |
| LMS | Learning Management System |
| SOAP | Simple Object Access Protocol |
| UM | User Model |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |

# 1 Introduction

Adaptive systems that deliver information (this includes adaptive hypermedia, recommender systems, information filtering, simulations, and to some extent even gaming) present their information in *units* at a time. Typical units are *pages* or *objects* (to be included in pages). The design of an adaptive application however is done in terms of (abstract) *concepts*, not in concrete information fragments. This deliverable explains how in GRAPPLE we make the link between *concepts* and *resources* (which can be pages, fragments, objects, or any other information unit that can be addressed or retrieved by querying some repository).

Figure 1 below shows a recent taxonomy of adaptation techniques in hypermedia, which is an update and extension of the taxonomies presented in [3] and [4]. The major difference between the new and old taxonomies is that "cosmetic" adaptation techniques (*adaptive presentation*) have been separated from techniques that really change the information content and the navigation possibilities. Connecting *concepts* with *resources* is only concerned with the "real" adaptation, thus the content adaptation and adaptive navigation techniques shown in the figure. (In the figure different types of arrows are used just to improve readability.)

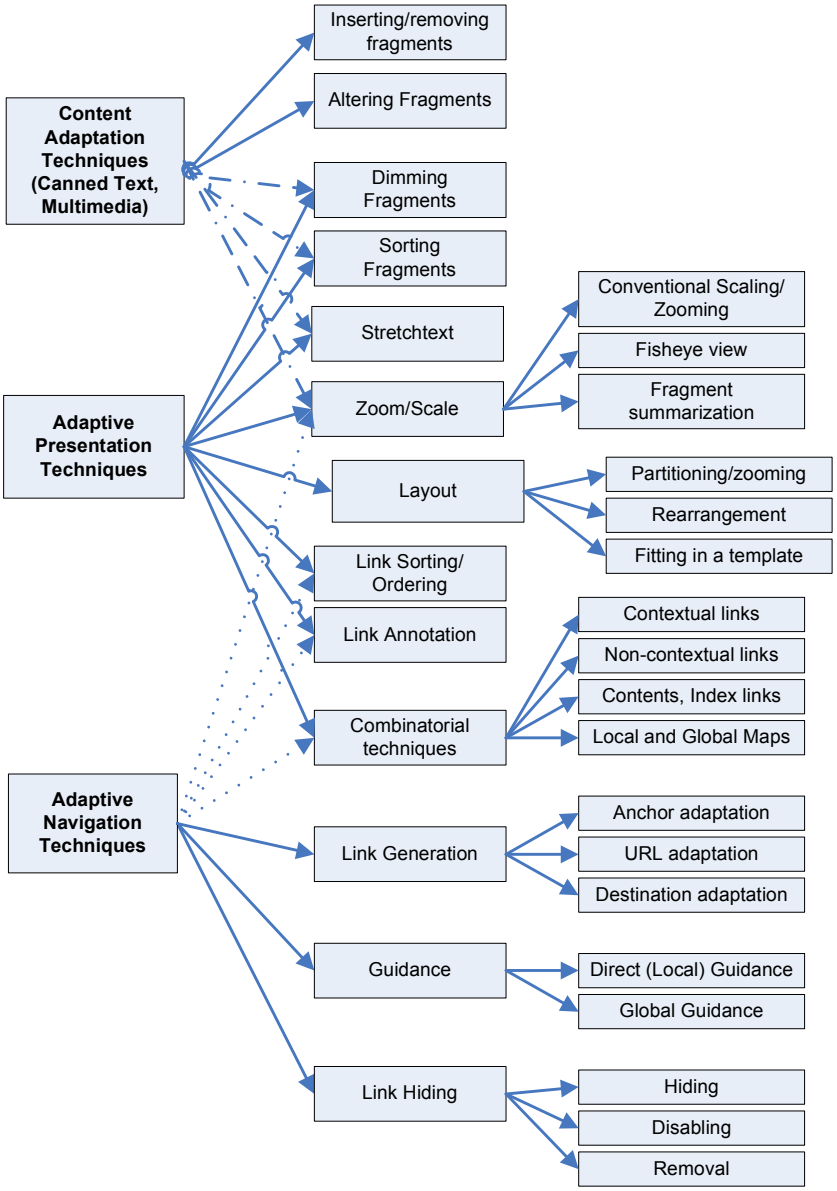


Figure 1: Taxonomy of adaptation techniques in hypermedia

The **GRAPPLE Adaptive Learning Environment** (or GALE) must "find" content for information that is addressed as concepts. This can happen in two ways: for *conditionally inserting fragments* in pages, and for retrieving *link destinations*. The *inserting/removing fragments* and the *altering fragments* techniques require finding resources (unless the fragments appear in-line on a page), and the *link generation,* and *guidance* techniques may also require a concrete implementation technique that we call *destination adaptation* (see Figure 1). The "translation" from *concept* to *resource* (or actually the *location* or URI of a resource) can be as simple as one-to-one, and can be as complex as posing a query (over the domain and user model) in order to compute or select the most appropriate resource. (Although often the URI may be an actual location and thus considered to be a URL we shall consistently use the term URI in this deliverable.)

# 2 Concepts and Resources in GRAPPLE

There are different possible approaches to linking *concepts* and *resources* in adaptive (educational) hypermedia systems:

- In Interbook [5] links on pages or in a navigation menu refer to *content*: They link to a fixed section or page of a course. On the server each page is linked to a number of concepts the page teaches something about. Links to concepts are also possible. Interbook typically shows a list of *background concepts* (prerequisites) and a list of *outcome concepts* (the concepts the page teaches something about). Accessing a concept can also be done through a "teach me" button that will generate a list of pages that (together) provide all the required prerequisite knowledge. Likewise in AHA! version 3 [2] links can refer to *concepts* or to *resources*. Internally AHA! would first translate the URI of a resource back to the name of the concept and then treat the link as a link to the concept.
- Another approach is to query a learning object repository (or a distributed federation of learning object repositories, or any other type of database). Such a query results in either a concrete page to be presented or in a possibly ranked set of results, linking to actual content (pages).
- A third possibility is to have links always refer to a *concept*, and to let the adaptation engine calculate which resource to retrieve and present. This follows the approach of the AHAM reference model [1]: accessing content is performed in two stages: a *page selector* translates concepts into identities of resources, and an *accessor* function retrieves the actual content. It is this approach we follow in GRAPPLE's adaptation engine GALE because it is most general. It allows *page selection* to be as simple as a one-to-one mapping between a concept and a page (in which case it could also be reversed to allow direct references to pages) or as complicated as the function of a search engine returning a ranked list of references to pages.

A key element in the GRAPPLE approach is that we make use of URIs to refer to both *concepts* and *content*. When creating a domain model an author has the option of uniquely identifying each piece of authored content with a concept. Selecting the appropriate content to be presented for a (higher level) concept can then completely be determined by the adaptation engine, and completely controlled by the author when designing the domain model and the conceptual adaptation (represented in the *Conceptual Adaptation Model* or CAM). However, it is equally possible to use a URI to express a query, so that when a concept is associated with a query it is up to the database or repository that handles the query to determine what content to return. In this case the author is only responsible for writing the query, not for how to answer it. Because both the input and the output of the *page selection* process consists of URIs it is possible to create a multi-stage process where the selection returns URIs that again refer to concepts so that the next page selection can be performed that may return the URI of a page or again of a concept, …

In Section 2.1 we describe how the GRAPPLE Adaptive Learning Environment GALE (see also deliverable D1.3a) performs *page selection*. Section 2.2 shows how these constructs can be expressed in the higher-level GRAPPLE Adaptation Language GAL (defined in deliverable D1.1a).

## 2.1 Concept to Resource translation in GALE

The translation of concepts to resources in GALE is inspired by AHA! version 3, but is much more generic. In AHA! every concept could be associated with a list of condition-URI pairs. The figure below shows a screen shot of the GUI element in AHA!'s Graph Author tool to define such condition-URI pairs.

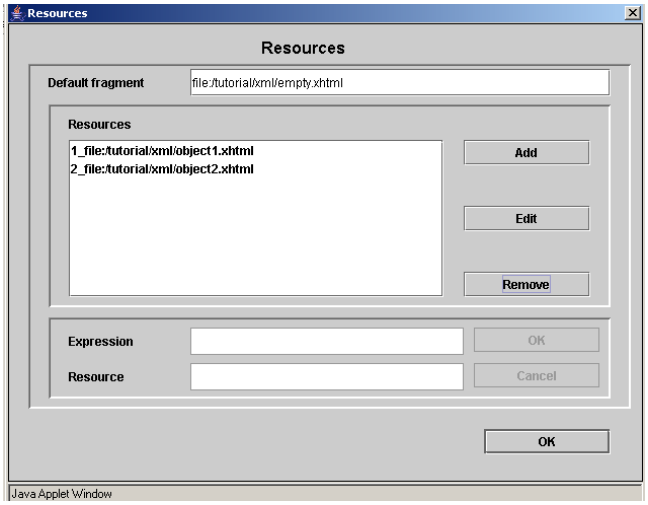


Figure 2: Resource Selection in AHA! 3's Graph Author

The condition (entered in the „Expression" field) is associated with a value for an attribute „showability". When the concept is accessed the expressions are evaluated to set the correct value of „showability", which is then used to select the appropriate URI. The Graph Author shields this implementation detail from the authors. An author simply writes condition-URI pairs.

In GALE we will keep the authoring simplicity through the authoring tools, while at the same time creating a more generic implementation. To this end the GALE engine uses the following approach:

- Requests for information that need to result in user model updates must be done through concepts. (It is possible to request resources directly, as done for instance to include fixed presentation elements like images, headers and footers, but these requests do not trigger any UM updates. Resources that are requested (and included) directly can still be adaptive, as may be done in a header for instance.)
- Each concept has a „resource" attribute that holds the URI currently associated with the concept (for the current user). When there is a one-to-one relationship between a concept and resource this is all that is needed.
- Like for any other attribute of GALE concepts the "resource" attribute also has a "default" expression. Default expressions can either indicate a value or an expression to be evaluated to initialize the attribute, or can be used to compute the attribute value each time it is needed, in case of volatile attributes (of which the value is not stored in UM). The syntax of such expressions is explained in deliverable D1.3a. GALE expressions are Java expressions in which one can refer to user model attribute values.

As an example, consider a course that is called "tutorial". It has a concept called "welcome" that can refer to two different pages: one for first-time visitors (say "readme.xhtml") and one for repeat visitors (say "welcome.xhtml). The "default" expression for tutorial.welcome should then be something like:

```
(${visited)==0?"aha:/tutorial/readme.xhtml":"aha:/tutorial/welcome.xhtml")
```

In this example we see the reference to `${visited}` which results in the value of the "visited" attribute of the current concept (for the current user). We assume here that the adaptation rules will ensure that "visited" represents the number of previous visits to the concept. If the concept has not been visited before the result is that the readme.xhtml file will be displayed, otherwise the welcome.xhtml file will be displayed.

Any Java expression can be used here, including expressions with method calls. The expression must return a string that can be interpreted as a URI. It is thus possible to extend GALE with modules that perform complex computations over the user model in order to decide which URI to associate with a concept. (This can be useful for some applications that wish to perform a page selection process that cannot be described in simple terms through the authoring tools that are currently being designed for GRAPPLE, or that cannot be expressed in the GAL language (of deliverable D1.2a)

After the URI of the resource has been determined the GALE engine can start its real work (retrieving the resource, updating the user model and adapting the page). The retrieved and adapted resource is eventually returned to the browser. For the browser it thus looks as if the presented content *is* a page with as URI the

requested concept. The browser is never passed the actual location (URI) of the page that is presented, only the concept(s). As a result, on the page all relative links to concepts simply work without having to do any manipulations to the URIs.

Because in GALE all requests refer to concepts using URIs and because the requests for content are initiated from GALE using URIs it is possible to have the computed URI refer to concepts as well as pages. It is thus possible to have a concept that refers to a chapter-concept associated with a list section-concepts from which the engine conditionally selects one. When the section is requested this is a new (HTTP) request to GALE that will call up the default expression associated with the section. This may result in selecting a page from a list of pages associated with the section. It is thus possible to traverse down the concept hierarchy, or to implement a „next“ button (or a „teach me“ button) that invokes a decision process to determine which page to show next. Note however that each of these requests for traversing the concept hierarchy is an “access event” and thus may cause UM updates.

Likewise an author can write pages (of a course) that contain conditionally included objects. The process for deciding which content to show for an object (that refers to a concept) is identical to the process for deciding on link destinations. Conditionally included objects (e.g. using the <object> tag in xhtml) cause the browser to generate HTTP requests and thus generate an event that causes UM updates.

Because GALE allows the use of arbitrary Java expressions (including method calls) it is possible to implement the most complex decision processes for linking concepts to resources. However, in practice, using GRAPPLE’s authoring tools and intermediate adaptation language GAL the selection will consist of simple expressions that are easy for authors to understand and create.

## 2.2 Concept to Resource translation in GAL

The GAL language is an intermediate language between the CAM and GALE. GAL is designed to provide the basic primitives to specify adaptive navigation. Authoring environments that specify adaptive navigable courses should be able to generate a GAL instance, which in its turn should be converted into adaptation rules that can be implemented by most adaptive engines. GAL is described in detail in deliverable D1.1a.

There is no specific construct for resource translation in GAL, but it can be simply done with typical GAL primitives. GAL Attributes represent literal values shown to the user. This can be text (fragments), but it can also be any URI-referable media types, e.g. pictures, videos or html pages. For concept to resource translation, typically the URI-referable media types are of importance. For determining which value (URI in the case of resource translation) an attribute should represent we can use a query which can refer both to the domain and user model. Moreover, we can use conditions to choose between different query values.

Let’s look at the previous example in Section 2.1. Consider the course that is called tutorial. It has a concept called “welcome” that can refer to two different pages: one for first-time visitors (say “readme.xhtml”) and one for repeat visitors (say “welcome.xhtml). In GAL terms we could write this as a Unit that contains the following attribute:

```
:hasAttribute [
                    :if [ :query
                        “ SELECT ?conceptVisits
                          WHERE
                              {
                                 :welcome :visited ?conceptVisits;
                                             :byuser $CurrentUser;
                                             :amount ?visits .
                                 FILTER (?visits > 0)
                               }”
                        ] ;
                   :then[:value “aha:/tutorial/welcome.xhtml” ] ;
                   :else[:value “aha:/tutorial/readme.xhtml”]
                 ]
```

In this example we see an attribute (:hasAttribute) that has a condition (if-then-else). In the “:if” clause we defined a query that queries the domain and user model. In this example we used the SPARQL[1] syntax for

[1] http://www.w3.org/TR/rdf-sparql-query/

the query, for now assuming that the domain and user modelled are expressed in RDF[2]. Note that this is just exemplary, other query formalisms and corresponding data models can be used as well in GAL.

The query looks at the concept ":welcome" which has a property ":visited" which points to a node denoted by the variable "?conceptVisits". This node is a concept to express the number of visits of concept "welcome" by the current user. We explicitly model here that the user for which we inspect the number of visits, i.e. the current user, must be $CurrentUser. The use of the $-sign indicates that this is a GAL variable. Note that the $-sign is not part of the SPARQL-language but is used by the GAL interpreter to substitute the value of the variable. Also note that "$CurrentUser" and "$CurrentApplication" are reserved variables that always exist in every GAL Unit. The variable "?conceptVisits" points to a node which should have a property ":amount" which should indicate the number of visits of the particular concept for this particular user. The variable "?visits" points to the number of visits by the user. In the FILTER clause it is then specified that we only include "?conceptVisits" if the number of visits is more than 0, so in case the user already visited the concept before.

If this query yields any results (i.e. user "$CurrentUser" did already visit concept ":welcome") we execute the ":then" clause and show the user the "welcome.xhtml" page. If user "$CurrentUser" is new, the else clause is executed and the user is shown the "readme.xhtml" page.

We could also use an example in which the resource can be directly found in the domain model, e.g.:

```
:hasAttribute [
        :query
          " SELECT ?nextURI
            WHERE
              {
                :welcome :visited ?conceptVisits;
                         :byuser $CurrentUser;
                         :hasseen ?seen .
                :welcome :haspage  ?page
                                   :hasurl ?nextURI;
                                   :advancedUser ?value.
                FILTER (?seen == ?value)
              }"
        ]
```

In this query we again look at the concept ":welcome" for user "$CurrentUser", but now instead of a counter we consider a boolean property ":hasseen" that records if the user has already seen the page or not. Moreover the ":welcome" concept has one or more properties ":haspage" and for such a page a URI for that page is defined (via ": hasurl") and a boolean attribute ":advancedUser" is defined which specifies if this page is for the advanced user (?value==true) or not (?value==false). We define :advancedUser as a user who has already seen the concept. We now select a page that is advanced or not by demanding only pages with an ":advancedUser" level that equals the ":hasseen" property for "?conceptVisits". This means that if the user has seen the concept he gets a page that is labelled for the advanced user, or if the user did not see the concept yet he gets a page that is labelled not being for an advanced user.

Note that this query can yield several results (depending on the domain model), e.g. if there are more than one pages labelled for the advanced user. If this is so, there is no guarantee which of those URIs will exactly be used by the engine. If this is not the desired behaviour the datamodel and queries should be designed such that the query yields exactly one result.

As shown there are several equivalent possibilities in which the same construct can be modelled in GAL. Of course, during implementation specific choices needs to be made how to translate CAM structures into GAL constructs. Fortunately, which choices are made is not relevant for the GAL language and equivalent choices will lead to the same behaviour in the Adaptive Engine(s).

[2] http://www.w3.org/RDF/

# References


1. De Bra, P., Houben, G.J., Wu, H., AHAM: A Dexter-based Reference Model for Adaptive Hypermedia, Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 147-156, Darmstadt, Germany, 1999.
2. De Bra, P., Smits, D., Stash, N., The Design of AHA!, Proceedings of the ACM Hypertext Conference, Odense, Denmark, August 23-25, 2006 pp. 133, and http://aha.win.tue.nl/ahadesign/, 2006.
3. Brusilovsky, P. Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction, 6 (2-3), pp. 87-129, 1996.
4. Brusilovsky, P. Adaptive hypermedia. User Modeling and User Adapted Interaction, Ten Year Anniversary Issue (Alfred Kobsa, ed.) 11 (1/2), pp. 87-110, 2001.
5. Brusilovsky, P., Eklund, J., Schwarz, E., Web-based education for all: A tool for developing adaptive courseware. Computer Networks and ISDN Systems (Proceedings of the 7th Int. World Wide Web Conference, 30 (1-7), pp. 291-300, 1998.