

GRAPPLE

3.2c Version: 1.0

Final Implementation of the Concept Relationship Type Tool

Document Type	Deliverable
Editor(s):	Christina M. Steiner (UniGraz), Alexander Nussbaumer (UniGraz)
Author(s):	Alexander Nussbaumer (UniGraz)
Reviewer(s):	Patrick Pekczynski (IMC), Luca Mazzola (USI)
Work Package:	WP3
Due Date:	M30
Version:	1.0
Version Date:	2010-12-30
Total number of pages:	37

Abstract: This report describes the final development of the Concept Relationship Type (CRT) Tool. A definition of the CRT and objectives and requirements of the CRT tool have already been described in the previous deliverable D3.2a and updated in D3.2b. The initial development of the CRT tool also has been described in deliverable D3.2b. The final implementation of the CRT tool and the final definition of the CRT, a user guide and some sample CRTs are given in this deliverable. The final implementation of the CRT tool is based on the findings of the previous deliverables and the updates of the CRT definition. The graphical user interface is described and a user guide is given, which explains how CRTs can be defined and which should provide help for the author. Finally the GUMF Mapping Tool is described, which allows for mapping user model variables in the GRAPPLE User Model Framework. This Tool has been implemented in the context of the CRT tool but is a separate tool in GAT.

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

Summary

In the previous deliverable D3.2a, Concept Relationship Types (CRT) have been defined and elaborated. A tool has been designed which allows defining CRTs. Based on this work the initial CRT tool has been implemented and reported in the deliverable D3.2b.

This report documents the final implementation of the CRT tool and the final specification of the CRT format. The report and implementation together form the deliverable D3.2c.

This deliverable not only reports on the updates, but also intends to give a comprehensive and self-contained specification of the final state of the CRT tool and format. However, it is documented where updates have been made since the initial implementation. Most important changes since D3.2b include updates on the format (such as introducing parameters) and the newly designed and developed GUMF Mapping Tool. The explanation of the meaning of CRTs and related background information can be found in D3.2a, but is not reported again.

CRTs have been technically specified using XML format. An XML schema has been defined that clearly defines which information is contained in a CRT and how this information is structured. An explanation of this information and structure is given in this report. Furthermore the software architecture of the CRT tool is described. An explanation of the graphical user interface is also included in this report.

Since there is a strong relation between the CRT tool and the other authoring tools (DM tool and CAM tool), a common software library has been created which offers functionalities to all tools. All tools together and the shared library used by all tools constitute the GRAPPLE Authoring Tools (GAT). This report describes how the CRT tool is included in GAT.

A user guide is given, which should help the author understand how CRTs can be defined and what the several sections of the CRT definition express. This is done in a step-by-step guide leading the author through the various sections and tabs of the CRT tool.

Finally, the GRAPPLE User Model Framework (GUMF) Mapping Tool is described which has been developed in the context of the CRT tool. This work has not been described in and was not required by the Description of Work (DoW). However, it turned out that authors need to have the possibility to map external user model data as provided by Learning Management Systems (LMS) to user model variables as defined in the CRT tool. Therefore, the GUMF Mapping Tool can be seen as a necessary extension of the CRT tool.

The initial implementation of the CRT tool has been demonstrated at the EC-TEL 2009 [3] conference and has been presented in a poster session at the ICCE 2009 [2] conference.

Authors

Person	Email	Partner code
Alexander Nussbaumer	alexander.nussbaumer@uni-graz.at	UniGraz

Table of Contents

SUMMARY	2
AUTHORS	2
TABLE OF CONTENTS	3
TABLES AND FIGURES.....	4
LIST OF ACRONYMS AND ABBREVIATIONS	4
1 INTRODUCTION.....	5
2 FINAL CRT FORMAT	5
2.1 Structure of the CRT format	5
2.2 XML Schema.....	9
2.3 Revisions	12
2.4 Unused sections of the CRT definition.....	12
3 IMPLEMENTATION AND INTEGRATION OF THE CRT TOOL	12
3.1 Implementation of the CRT tool	12
3.2 Integration of the CRT tool with the GAT shell and Web Service.....	13
4 GRAPHICAL USER INTERFACE OF THE CRT TOOL.....	14
5 USER GUIDE AND IMPORTANT CRTS.....	17
5.1 User Guide on Managing and Defining CRTs	17
5.1.1 Managing CRTs.....	17
5.1.2 Defining a CRT	19
5.2 Some Important CRTs as Examples	27
6 GUMF MAPPING TOOL	28
6.1 Purpose of the GUMF Mapping Tool.....	28
6.2 Design and Specification of the GUMF Mapping Tool	28
6.3 User Guide to the GUMF Mapping Tool.....	30
7 CONCLUSION	33
REFERENCES	33

DOCUMENT CONTROL ERROR! BOOKMARK NOT DEFINED.

8 APPENDIX **33**

8.1 CRT Examples..... **33**

Tables and Figures

List of Figures

Figure 1: CRT tool architecture designed as Model-View-Controller design pattern. 13

Figure 2: CRT tool integrated in the GAT shell 14

Figure 3: CRT tool: Creating a new CRT..... 18

Figure 4: CRT tool: Defining properties for a new CRT..... 18

Figure 5: CRT tool: Opening a CRT 19

Figure 6: CRT tool: Defining the visual representation and the structure. 20

Figure 7: CRT tool: Defining a CRT with an undirected structure and one single socket. 21

Figure 8: CRT tool: Pedagogical meaning and parameters of the CRT..... 22

Figure 9: CRT tool: Defining the code of the CRT..... 23

Figure 10: CRT tool: Defining the user model variables of the CRT. 24

Figure 11: CRT tool: Defining the constraints of the CRT. 25

Figure 12: CRT tool: Defining the usage of domain model relations for this CRT. 26

Figure 13: CRT tool: Insight to the XML representation of the current CRT. 27

Figure 14: GUMF Mapping Tool: Launching the tool. 30

Figure 15: GUMF Mapping Tool: The tool before initialisation. 31

Figure 16: GUMF Mapping Tool: Existing rules have been loaded from GUMF..... 32

Figure 17: GUMF Mapping Tool: A selected rule. 32

List of Acronyms and Abbreviations

CAM	Conceptual Adaptation Model; also called: Adaptive Story Line; Adaptation Strategy
CRT	Concept Relationship Type(s); also called: (Pedagogical) Relationship Type(s)
DM	Domain Model
GAL	GRAPPLE Adaptation Language
GAT	GRAPPLE Authoring Tools
GALE	GRAPPLE Adaptive Learning Environment
GUMF	GRAPPLE User Model Framework
GRAPPLE	Generic Responsive Adaptive Personalised Learning Environment
GUI	Graphical User Interface
LMS	Learning Management System
UM	User Model

1 Introduction

The concept relationship type (CRT) tool constitutes a tool to be implemented in the GRAPPLE environment for defining relationship types on a conceptual level. The CRT tool is working together with the domain model (DM) on one hand and with the conceptual adaptation model (CAM) on the other hand. Concept relationships types are defined in order to be used in interrelation with and referring to the DM in the context of the CAM, which uses the output of the CRT tool for defining actual adaptation strategies. The output of the CRT tool is a set of CRTs, whereby each CRT can be instantiated by the CAM authoring tool. In the CAM authoring tool adaptation strategies can be defined by connecting concepts with relationships which are defined in the CRTs.

By using Concept Relationship Types (CRT) a teaching strategy can be created. With CRTs concepts from the domain model (DM) can be connected to create a pedagogically meaningful strategy which is defined through the Conceptual Adaptation Model (CAM). This model determines how and when learning resources related to these concepts should be presented to the learner. A detailed elaboration on CRTs and how they are related to the DM and CAM is given in D3.2a.

The CRT Tool has been implemented in order to empower the course author to define CRTs more easily. This report describes the CRT tool from a technical perspective, which includes the XML representation of CRTs, the software architecture and the graphical user interface (GUI).

Section 2 describes the final specification of the CRT format. First, it is explained which information is contained in a CRT and how this information is structured. Second, a technical specification of the CRT format is given by means of an XML schema. Finally some revisions made since the last deliverable on CRTs are listed.

In section 3 the software architecture of the CRT tool is described. Basically, the software architecture of the CRT tool consists of three parts which are the graphical user interface, the application logic and the data model. Since the CRT tool is integrated within the GRAPPLE Authoring Tools (GAT) shell, this integration is also described (in a technical sense).

The graphical user interface is described in section 4. With the help of several screenshots it is shown how CRTs can be defined. A detailed user guide is given in section 5 which guides the author through the authoring process of the CRT definition.

The last section (section 6) describes the GUMF Mapping Tool. Since this is a separate tool in addition to the CRT tool, its design, functionalities and user guide are described together in a separate section. The goal of the GUMF Mapping Tool is to map external user model data as provided by Learning Management Systems (LMS) to user model variables as defined in the CRT tool.

2 Final CRT Format

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

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

In this section the common explanation of the CRT format and structure is given, followed by the XML schema definition. Finally it is outlined which changes have been made to the CRT definition since the last deliverable D3.2b. Some examples of CRT definitions can be found in the appendix.

2.1 Structure of the CRT format

For compatibility reasons with the other model formats of the GRAPPLE Authoring Tools (GAT), a common wrapper format has been defined. Each model, such as CAM, DM, and CRT, has a wrapper structure, which allows for a unified storing and retrieving from and to the database on the web. The web service which accepts models for storing and delivers requested models from the database does not need to know which type the models are.

The following XML code outlines the common structure of GAT models. Each model has a header part, where metadata of the model is located. Metadata include title, common description, creation and update time and date (timestamp), author, authorisation and the UUID of the model.

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

The CRT definition can be found in the body part of the model. It is partitioned into eight areas, where each one is specified as a more or less complex element in the XML specification. The following piece of XML code depicts this structure with the eight elements. The numbers (as comment) before each element refer to the sub-sections below.

```
<crt>
  <!-- (1) --> <crtdialect>...</crtdialect>
  <!-- (2) --> <comment>...</comment>
  <!-- (3) --> <parameter>...</parameter>
  <!-- (4) --> <visualrepresentation>...</visualrepresentation>
  <!-- (5) --> <crtsockets>...</crtsockets>
  <!-- (6) --> <adaptationbehaviour>...</adaptationbehaviour>
  <!-- (7) --> <constraints>...</constraints>
  <!-- (8) --> <associateddmrelations>...</associateddmrelations>
</crt>
```

(1) CRT dialect

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

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

(2) Comment

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

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

(3) Parameter

The parameter field allows for defining parameters which can be used to assign fixed values to a variable (parameter) either in the CRT tool or later in the CAM tool. The value field can be left blank or filled with a certain value. In both cases the value can be set or changed in the CRT tool or the CAM tool. An example looks like this:

```
<parameter name="level" type="integer">80</parameter>
```

This parameter can be used in the code fragment as a variable. In the GALE code, this parameter can be used as a variable which gets its value either in this specification or later in the CAM tool. This has the advantage that authors do not have to use fixed values in the code but can use parameters and set the value later. One can either assign a value to the variable in the CRT tool or individually for each CRT instance in the CAM tool. When deployed to GALE the value of the parameter is used during the code interpretation.

Possible data types for the type attribute are "integer", "float", "boolean", and "string". If there is more than one parameter defined, the parameter definitions are encapsulated in an parameter XML field:

```
<parameters>
  <parameter name="level" type="integer">80</parameter>
  <parameter ....>
</parameters>
```

(4) Visual representation

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

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

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

(5) CRT Sockets

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

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

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

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

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

(6) Adaptation behaviour

The definition of the adaptation behaviour is the central part of the CRT definition. It defines the pedagogical meaning of the CRT in a machine-readable way for the interpretation within the GALE engine. The definition of the adaptation behaviour is done through a piece of code written in the GRAPPLE Adaptation Learning Environment (GALE) language. The GALE code is inserted into the XML code as it is (using CDATA). The specification of the GALE code (syntax and meaning) is done in WP1. The following example shows how the GALE code is inserted. For future purposes, the definition also allows the usage of another kind of code. This behaviour is implemented by the changing of the relevant attribute in the type attribute in the tag.

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

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

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

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

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

An optional child of an *umvariable* XML element is the *location* element, which indicates the external variable mapped to this user model variable.

```
<umvariable>
  ...
  <location>
    <web>1</web>
    <remotecourse>
      <remotecoursename>2</remotecoursename>
      <resource>
        <resourceuniqueid>4</resourceuniqueid>
        <resourcename>3</resourcename>
      </resource>
    </remotecourse>
  </location>
</umvariable>
```

The location field is currently not used by any GRAPPLE component.

(7) Constraints

In this section three types of restrictions can be defined regarding CRTs. First, one can specify if loops in the CAM are allowed in this CRT. For example, it would not make sense to allow loops within a prerequisite CRT. Secondly, one can restrict which concepts can be put into a socket.

That is why a triple of socket UUID, attribute name and attribute value is used to define that the socket with the given UUID can only contain concepts that have a specific attribute value for the provided attribute name.

Thirdly, one can restrict which CRTs are not allowed in the "neighbourhood" when CRTs are used in the CAM. Concepts respectively sockets containing concepts are connected by CRT instances in the CAM, however, with this restriction it can be excluded that certain CRT instances are connected to the same socket. This is done by naming user model variables which appear in CRTs that are not allowed to be in the "neighbourhood". The following example shows how to define these restrictions.

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

Constraints are currently not used by any GRAPPLE component.

(8) Associated domain model relations

Associations to relations between concepts used in the domain model can be exploited. For inserting CRTs existing structures of the domain model can be used along with which CRTs can be laid automatically. The following example shows how relations between concepts in a domain model are used for this CRT.

```
<associateddmrelations>
  <relationshipiptype>is_a</relationshipiptype>
  <relationshipiptype>part_of</relationshipiptype>
</associateddmrelations>
```

Domain relations are currently not used by any GRAPPLE component.

2.2 XML Schema

This section contains the XML schema specification of the CRT XML format. CRTs created by the CRT tool are valid against this schema specification. This specification is supposed to be used by all GRAPPLE tools and components which read CRTs. Examples of CRTs which implement this specification can be found in the appendix.

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://grapple-project.org/GAT/"
  xmlns="http://grapple-project.org/GAT/"
  elementFormDefault="qualified">

  <!-- (A) The overall structure and the model header -->

  <xs:element name="model">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="header">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="creationtime" type="xs:integer" minOccurs="1" maxOccurs="1"/>
              <xs:element name="updatetime" type="xs:integer" minOccurs="1" maxOccurs="1"/>
              <xs:element name="modeluuid" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="authoruuid" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="authorisation" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="modeltype" type="modeltypetype" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```

</xs:element>
<xs:element name="body">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="crt" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:all>
            <xs:element ref="crt dialect" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="comment" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="parameter" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="parameters" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="visualrepresentation" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="crtsockets" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="adaptationbehaviour" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="constraints" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="associateddmrelations" minOccurs="1" maxOccurs="1"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:simpleType name="modeltypetype">
  <xs:restriction base="xs:string">
    <xs:enumeration value="crt"/>
  </xs:restriction>
</xs:simpleType>

<!-- (B) The eight elements of the CRT definition -->
<!-- (B.1) the CRT dialect -->
<xs:element name="crt dialect" type="crt dialecttype"/>

<xs:simpleType name="crt dialecttype">
  <xs:restriction base="xs:string">
    <xs:enumeration value="CRT"/>
  </xs:restriction>
</xs:simpleType>

<!-- (B.2) the comment field -->
<xs:element name="comment" type="xs:string"/>

<!-- (B.3) the parameter field -->
<xs:element name="parameters">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="parameter" type="parameterType"/>
<xs:complexType name="parameterType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="type" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!-- (B.4) the visual representation -->
<xs:element name="visualrepresentation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="colour" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="shape" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- (B.5) the sockets -->
<xs:element name="crtsockets" >
  <xs:complexType>
    <xs:sequence>

```

```

<xs:element name="socket" minOccurs="1" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="uuid" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="mincardinality" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="maxcardinality" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<!-- (B.6) the adaptation behaviour -->
<xs:element name="adaptationbehaviour">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="code" type="codeType" minOccurs="1" maxOccurs="1"/>
      <xs:element name="usermodel" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="umvariable" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="codeType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="type" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="umvariable">
  <xs:complexType>
    <xs:all>
      <xs:element name="umvarname" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="socket" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="public" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="persistent" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="type" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="range" type="xs:anyType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="default" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="location" type="xs:anyType" minOccurs="0" maxOccurs="1"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<!-- (B.7) constraints -->
<xs:element name="constraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="loopsallowed" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="attributeconstraints" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="umvariableconstraints" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="attributeconstraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="attrconstraint" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="socketid" type="xs:string" minOccurs="1" maxOccurs="1"/>
            <xs:element name="attributename" type="xs:string" minOccurs="1" maxOccurs="1"/>
            <xs:element name="requiredvalue" type="xs:string" minOccurs="1" maxOccurs="1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

</xs:element>
<xs:element name="umvariableconstraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="umvariablename" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- (B.8) associated domain model relations. -->
<xs:element name="associateddmrelations">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="relationshipstype" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

2.3 Revisions

There have been updates to the CRT format since the last deliverable on CRTs (D3.2b). The code in the section "adaptive behaviour" now has a type. Currently only the type "gale" is supported but this field can be defined freely. Any type of code can be used as long as GALE can handle it.

Additionally the possibility to define parameters has been introduced to the CRT tool. A new XML element has been included in the CRT format definition that allows for defining a parameter. Such a parameter can be assigned with values either in the CRT tool or later in the CAM tool individually for each CRT instance. This parameter is referenced in the GALE code and the value resolved by the GALE engine.

The colour element for each socket has been removed, since it is sufficient to specify the colour for the whole CRT. It did not seem useful for the author to have CRTs with different colours because the main purpose of the coloured CRTs is that they can be visually distinguished from each other.

2.4 Unused sections of the CRT definition

As already mentioned in section 2.1 there are some parts in the CRT definition which are currently not used by any GRAPPLE component. This results from the fact that the original planning went beyond the actual development. Some fields are not used yet, however, they are kept in the CRT definition in order to allow further development (before or after the project has ended). The unused parts are:

- all constraints definitions for limiting the instantiation possibilities in the CAM tool
- domain relation definitions for allowing the (semi-)automatic creation of CAM models
- the location part in the user model variable section, which refers to external user model variables

3 Implementation and Integration of the CRT tool

The basic design of the CRT tool has already been outlined in D3.2a. It has been described that the tool has a user interface where the author can create, edit and modify CRTs. CRTs should be stored on the server side using a web service and a database behind the web service.

For a more efficient implementation the common functionalities of the DM tool, the CRT tool and the CAM tool have been integrated into the GRAPPLE Authoring Tools (GAT) shell. The functionalities of the GAT shell are provided to all of these tools. The most important functionalities are providing a visual container with a menu bar and handling the creation, loading and saving of data models (see for example Figure 5).

3.1 Implementation of the CRT tool

From a technical point of view the CRT tool consists of three parts, which are the user interface, the application logic and the CRT data model (see Figure 1). This design follows the Model-View-Controller

(MVC) design pattern, where the user interface is the view, the application logic is the controller and the CRT data model is the model.

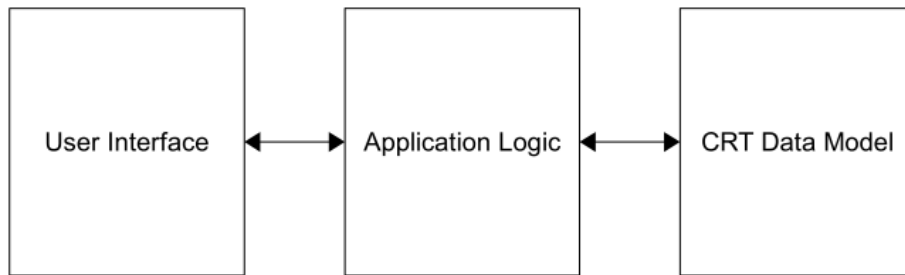


Figure 1: CRT tool architecture designed as Model-View-Controller design pattern.

The user interface (UI) is the place where the author can create and modify CRTs. Basically it is a graphical representation of the CRT format which is described in section 2. All the data fields defined in the CRT specification can be accessed through this UI of the CRT tool. Screenshots and an explanation of the UI are provided in sections 4 and 5.

The application logic is the component that handles the events from the UI. Since parts of the CRT are related or depend on each other, modifications of these parts may affect other parts of the CRT. The application logic is also responsible for handling loading and saving CRTs.

The CRT data model is the internal representation of the CRT. It is structured according to an object-oriented approach. The data model is also responsible for converting the CRT to and from the XML representation. Since there are several classes representing specific parts of the CRT, each of these classes is responsible for the respective conversion to and from the XML representation.

The implementation of the CRT tool has been done using the Adobe Flex [1] framework. Flex offers good support for web-based and graphical applications. The definition of the user interface is represented as an XML file and separated from the other parts of the application. The Flex compiler processes this XML definition of the UI and creates a graphical and interactive user interface of it. Flex projects are compiled into Flash applications which can be executed by web browsers that have the Flash plugin installed.

3.2 Integration of the CRT tool with the GAT shell and Web Service

The GAT shell is a Flex library which can be integrated into the tools. It provides a graphical container, a menu and the functionality for loading and saving models from and to the web service containing the GAT database. For persistence reasons, the models have to be saved on the web in a database which can be accessed through web services. Figure 2 depicts how the CRT tool is integrated with the GAT shell.

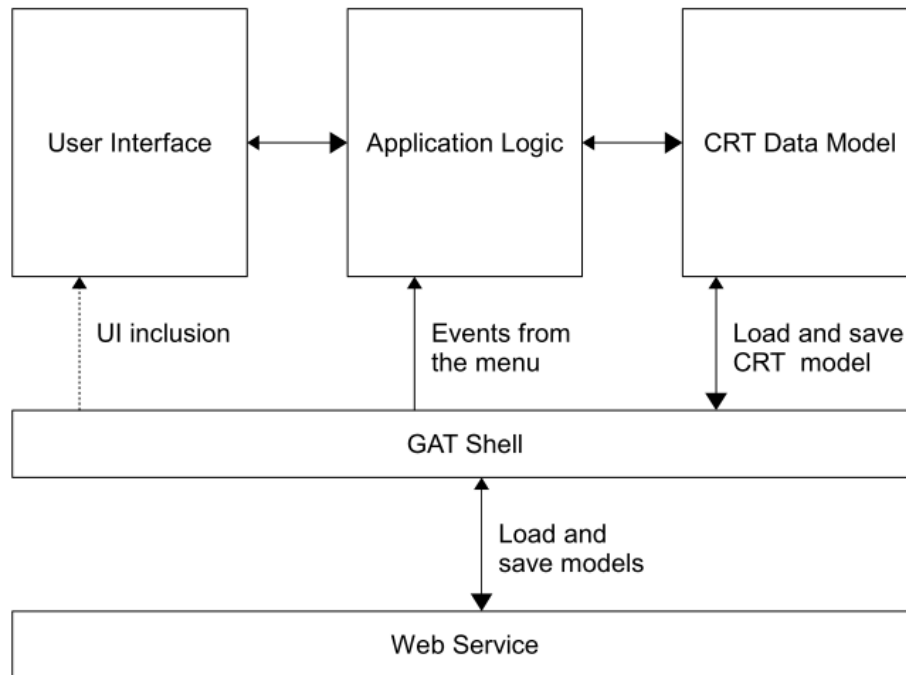


Figure 2: CRT tool integrated in the GAT shell

The GAT shell provides the menu from which events are raised if a user activates menu items. Consequently, they have to be sent to the CRT tool. The most important events are "open model", "save model", and "save as". These events are received by the application logic which activates the XML (de)serialisation of the internal CRT. The XML code is sent to or retrieved from the GAT shell which is responsible for saving and loading the models to and from the web service.

4 Graphical User Interface of the CRT tool

The graphical user interface (GUI) is an important part of the CRT tool since the purpose of this tool is to empower authors to define CRTs with the help of a graphical interface. The requirements of the CRT tool clearly point into the direction of an easy to understand and usable interface for authors. The GUI in this final implementation is the version which has continuously been updated since the initial version. The GUI provides the possibility to edit all parts of the CRT body so a complete CRT can be defined. The header part (title, description, etc.) cannot be edited, they are just displayed.

The information needed to specify a CRT includes:

- general information of CRTs, such as name and description, comment, creation time and author (cannot be edited, just displayed)
- information how an instance of a CRT should be visually represented in the CAM
- the structure of the CRT defined by so-called sockets
- the properties of the socket (source and target in case of a directed structure)
- freely defined parameter which can be used to assign values individually for CRT instances
- the GALE code which defines the adaptation behaviour in terms of instruction for GALE engine
- the user model variables accessed by the current GALE code
- constraints, such as the information if sequences of CRT instances may form a loop (which would not make any sense for prerequisite relationships) or if concepts with specific attributes are excluded (not exploited by other GRAPPLE components)
- relations to domain model relations (not exploited by other GRAPPLE components)

According to these parts of the CRT format specification there are several sections in the GUI where the respective parts in the CRT can be defined. These sections are structured with tabs which the author can use to switch between the different parts of information.

In the following part the tabs are explained using screenshots for each of them. The GUI is also explained in the next section (User Guide), where the referenced screenshots can be found.

Tab 1: General

In the first tab ("General", Figure 6 and Figure 7), the following CRT properties are displayed and can be specified:

- Title: The title of the CRT is displayed (cannot be edited)
- Description: The description of the CRT (which should be a general verbal description) is displayed (cannot be edited)
- Visual Representation: The visual representation consisting colour and shape, is specified; this defines how a CRT is represented in the CAM tool.
- Structure: The socket structure and the properties of sockets can be specified. Sockets are container where concepts can be placed in the CAM tool. The properties of the sockets include the name of the socket and minimum and maximum number of concepts which can be placed in a socket. One can also choose between a directed or undirected structure:
 - A directed structure means that there is a source and a target socket which are used by the CAM to define a directed structure on concepts.
 - An undirected structure means that concepts in sockets are not directed. Any number of sockets is allowed for this mode. In most cases only one socket is used.

Tab 2: Meta Info

In the second tab ("Meta Info", Figure 8), the following CRT properties can be specified:

- Comments: The author can describe the pedagogical meaning as free text, which is not processed by GRAPPLE. This information is useful if authors use existing CRTs and want to know which pedagogical meaning the selected CRT has.
- Creation and modification time: The time is automatically stored in these fields (cannot be changed by the user however, the user should know when a CRT was created and updated the last time)
- Author: the author of the CRT is automatically stored (cannot be changed by the user however, the user should know by whom this CRT was originally created). A limitation regarding the appearance of this field is that the UUID of the author is displayed instead of the user name.
- Parameters: Parameters can be defined in terms of parameter name, parameter type and parameter value. Possible types are "float", "integer", "string", and "boolean". An arbitrary number of parameters can be defined.

Tab 3: Code

In the third tab ("Code", Figure 9), the following CRT properties can be specified:

- Type: The type field specifies the type of the code. Currently, only type "GALE" is useful, since this is the only type which can be interpreted by GALE. For the sake of flexibility, an arbitrary string can be inserted in this field, as long as this type can be interpreted by GALE.
- Code: The code defines the adaptive behaviour (the meaning of the CRT on a formal level). This is a piece of code provided in a language understandable by the adaptation. This piece of code will be interpreted by GALE.

Tab 4: User Model

In the fourth tab ("User Model", Figure 10), the user model variables accessed by the code can be defined. All user model variables used in the code fragment have to be defined in this tab (except if they are formally defined in the code). UM variables can be added or removed in and from the list on the left hand side. For each variable the following properties can be specified:

General information:

- Name: The name of the user model variable. This has to be the same name as in the code as it is used to match it.
- Socket: At runtime user model variables are associated with concepts. However, at CRT level there are no concepts but only sockets where concepts will be included in the CAM tool. In order to associate user model variables with concepts, the user model variable is associated with a socket and later on all concepts inserted in this socket are associated with this user model variable. That is why the socket has to be specified to which this UM variable applies.
- Storage type:
 - public: This field defines whether GALE needs to submit updates of this user model variable to GUMF or not. If the user model variable is public, than all concepts associated with this user model variable are published to GUMF and also the respective value is published.
 - internally persistent: This field defines whether the value of this user model variable is stored internally by GALE or if it is computed or retrieved on GUMF. If a user model variable is defined as internally persistent, the GALE keeps the user model variable and its value.

Value information

- Type: The type of the UM variable can be defined by choosing it from the select box. It can be integer, float, string, or boolean. The type determines which values can be expressed by this user model variable.
- Range: The range limits the range of values according to the specified type. The range field is optional.
- Default: This field specifies the default value of the user model variable. The default value is optional.

Location

- The fields in the location part (specifying the external source for the variable) are currently not used by other GRAPPLE components (GALE).

Tab 5: Constraints

In the fifth tab ("Constraints", Figure 11), constraints can be defined limiting the usage of CRTs in the CAM tool. The following constraints can be specified. However, all constraints fields are currently not used by other GRAPPLE components (CAM tool). Nevertheless, the constraints information is contained in the XML representation of the CRT and stored in the GAT database together with the CRT. This allows a future version of the CAM tool to process this information once it is able to.

- Loops: This flag defines if it is allowed to create loops in the CAM tool with this CRT. This is currently not used by other GRAPPLE components (CAM tool).
- Attributes: This table defines what concepts can be added to a given socket. For example, it can be specified that only concepts can be added which have the value "en" for the attribute "language". Attributes are taken from the properties of the concepts (see Domain Model Specification). This is currently not used by other GRAPPLE components (CAM tool).
- User model variables: CRTs accessing user model variables from this list are not allowed to be 'neighbours' in the CAM. This is currently not used by other GRAPPLE components (CAM tool).

Tab 6: Domain Relations

In the sixth tab ("Domain Relations", Figure 12), the relations to domain models can be specified. However, domain relations are currently not used by other GRAPPLE components. Nevertheless, the domain relations

information is contained in the XML representation of the CRT and stored in the GAT database together with the CRT. This allows a future version of the CAM tool to process this information once it is able to.

- Domain relations: This list defines what relations between concepts in the domain model can be exploited for an automatic generation of a CAM model. Along the given relations in this list, CRTs are automatically created and concepts are inserted which are connected with these domain relations. This is currently not used by other GRAPPLE components (CAM tool).

Tab 7: Debug

In the seventh tab ("Debug", Figure 13), the CRT is represented in XML format. Originally, this tab was only used for development purposes. However, it turned out that there is some practical usage for this tab. Authors can easily exchange parts of the CRT Definition without doing it manually. Secondly, in case of bugs, this information can be used to create bug reports.

- The current CRT is represented in XML format in the text area. Modifications can be made and stored back to the CRT tool by pressing the "Save to Pedagogical Relation Tool".

5 User Guide and Important CRTs

This section first gives a short step-by-step guidance how CRTs can be defined using the CRT tool. This includes the management of CRTs in terms of opening and saving CRTs, as well as the definition of CRTs. Secondly a collection of important CRTs is listed, also in order to demonstrate to users how to implement a CRT. In order to define CRTs some prerequisite knowledge is needed, which is not provided in this deliverable but in other deliverables (see below for references).

5.1 User Guide on Managing and Defining CRTs

The notion of "Concept Relationship Type" has been changed in the user interface in order to better express the meaning of CRTs. They are now called "Pedagogical Relationship Types".

Prerequisite Knowledge

In order to define new CRTs, knowledge about the Conceptual Adaptation Model (CAM) and GALE is required, which is described in other deliverables: deliverable D3.3c explains the CAM and how CRTs are included in this kind of models and which role they play. Especially section 5 gives a good description how a CAM is created using CRTs. The meaning of CRTs regarding adaptive behaviour and pedagogical approaches has been described in deliverable D3.1a. Most importantly, the syntax of the GALE code and how adaptive behaviour can be achieved with code fragments in CRTs is described in deliverable D1.1b. All this knowledge is needed to define useful and proper CRTs.

5.1.1 Managing CRTs

Creating a new CRT

In order to create a new CRT, the first step has to be done in the GAT shell. By default, tabs and boxes regarding CRTs are not visible in the GAT Shell. For enabling these graphical elements, the radio menu item "Advanced Mode" in the menu item "Tools" has to be selected (see Figure 3). If this option is selected, the CRT tabs and boxes appear in the GAT shell. In the "Welcome" tab new CRTs can be created by clicking on the "Create a Pedagogical Relationship Type" box.

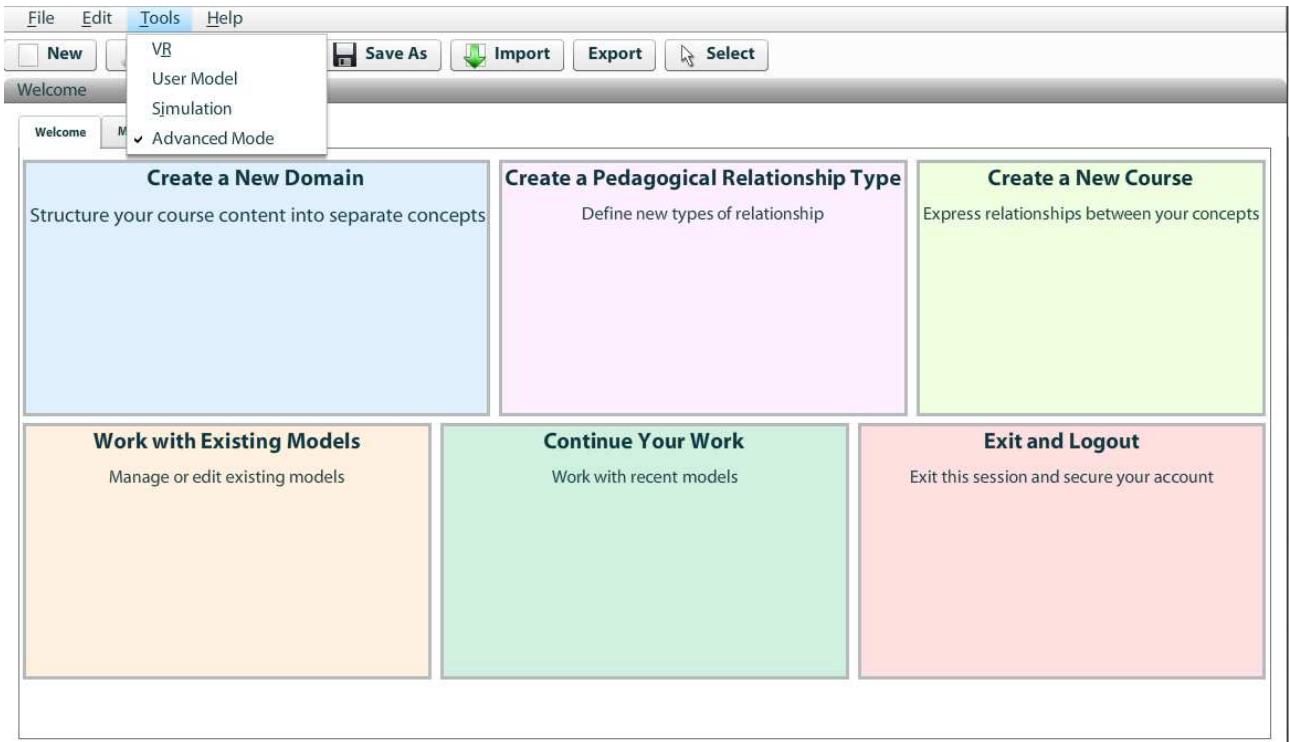


Figure 3: CRT tool: Creating a new CRT.

Properties of a new CRT

If a new CRT is being created, a dialogue appears (Figure 4) where the general properties of the CRT can be set. The title is the name of the CRT and the description should contain a short and general explanation of the CRT. The pedagogical meaning should not be entered here. The access rights (*public*, *read only*, or *private*) property indicate whether other users can modify the CRT, whether they can only read the CRT or whether the CRT is being displayed to them at all. If this information has been provided, the user needs to press the "Start Editing" button, which opens the CRT tool.

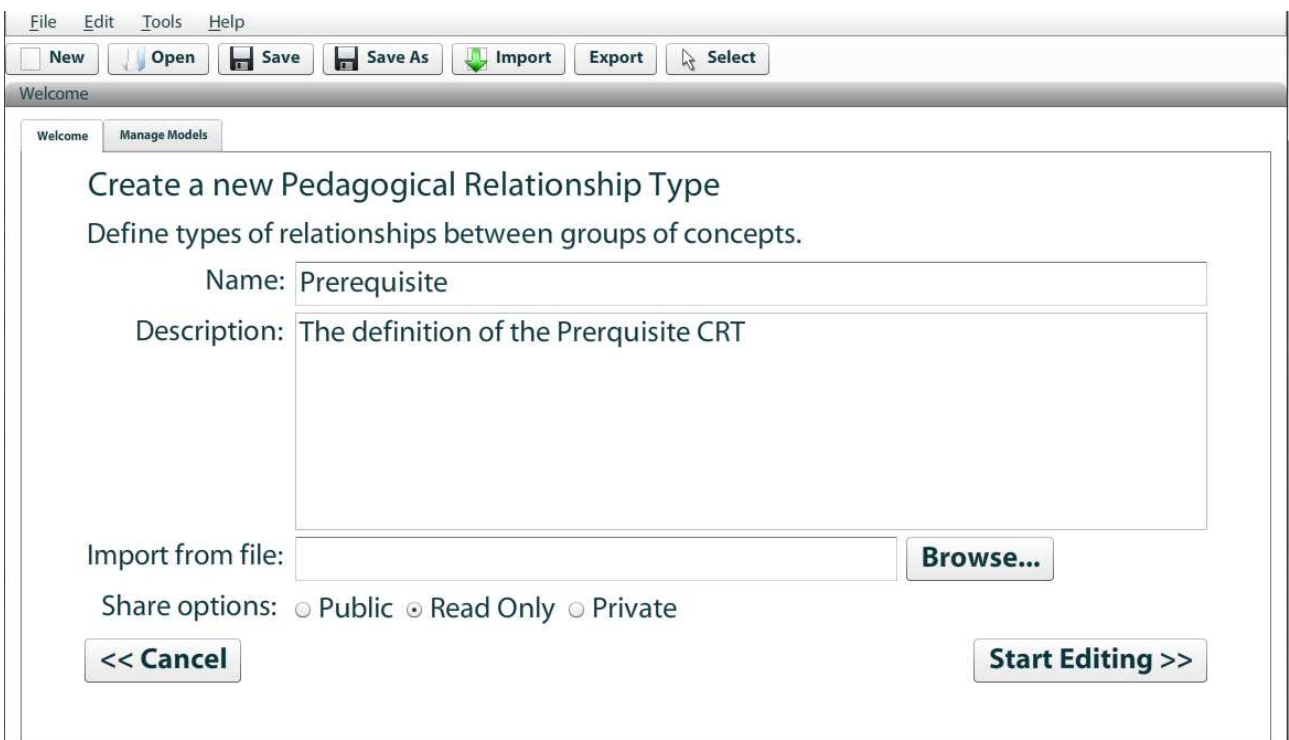


Figure 4: CRT tool: Defining properties for a new CRT.

Opening an existing CRT

Opening an existing CRT is done in the "Manage Models" tab in the GAT Shell (Figure 5). A list of existing CRTs will be displayed in the "Manage Models" tab where the user can choose a CRT and open it by left-clicking on the "Start Editing" button. In this tab it is also possible to change properties (title, description, access rights) of a CRT. CRTs can also be cloned, exported or deleted.

Saving CRTs

As described in section 3, CRTs can be loaded and saved to a web service storing the CRTs in a database. This functionality is provided by the GAT shell. In the CRT tool this can be done by using the menu items in the file menu of the GAT shell as soon as a CRT has been opened. Saving a CRT is performed by using the "Save" or "Save as" menu item.

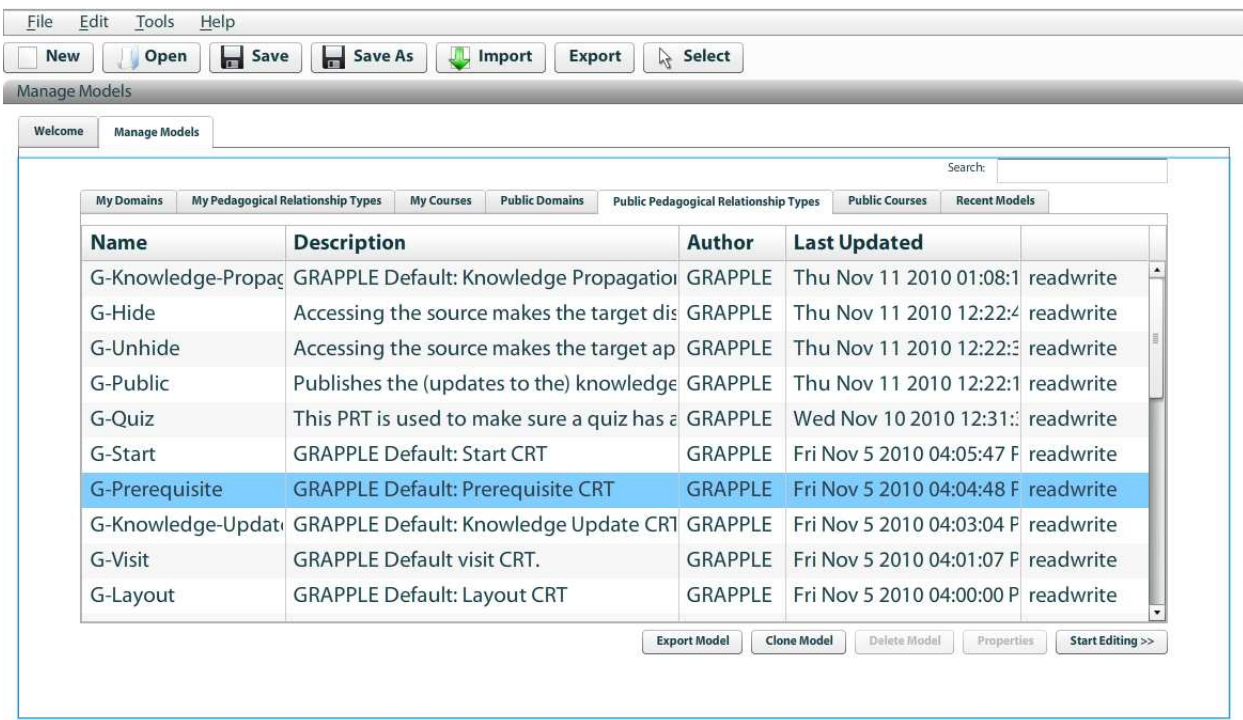


Figure 5: CRT tool: Opening a CRT

5.1.2 Defining a CRT

General Tab

Information on title and description

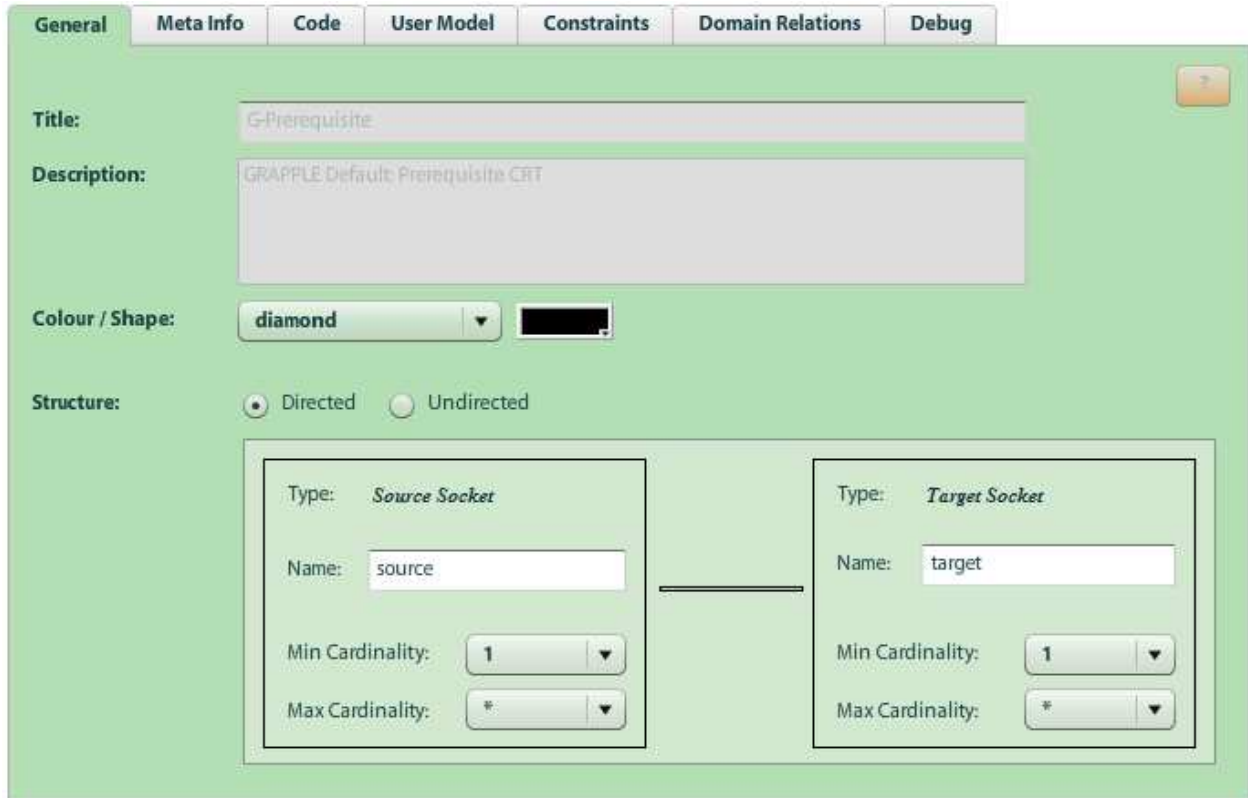
As soon as the CRT tool has been opened, the first tab "General" is visible (Figure 6). At the top of this tab, the title and description of the CRT is visible, as it has been defined in the property dialogue. It cannot be changed in the CRT tool. However, in the "Manage Models" section of the GAT shell, there is a button to modify properties. In order to modify the properties, the CRT tool has to be closed, then the properties can be changed and then the CRT has to be opened again.

Defining the visual representation

In the second part the visual representation can be specified. In the CAM tool, CRTs are depicted graphically and represented using a specific colour and a specific symbol. This colour and symbol can be changed by selecting the symbol in the pull down list and the colour in the colour picker. The selected colour is immediately displayed in the structure below (next paragraph).

Defining the structure of the CRT

At the bottom of this tab the structure of the CRT can be specified. This is one of the most important parts of the CRT definition. The basic element of a CRT is the socket, which is a container where domain concepts (of the domain model) can be inserted as soon as the CRT is inserted in a CAM. In the CRT the properties of the sockets are defined. First, the name has to be specified, which is used to be addressed in the code fragment. Second, the minimum and maximum number of concepts to be inserted in the respective socket is determined. Third, the structure of the CRT is defined: either a directed structure or an undirected structure can be chosen. In case of a directed structure the CRT includes exactly two sockets, where one of them is the source and the other is the target socket. The meaning of source and target sockets however, is defined in the code part. If an undirected structure is chosen, the CRT can contain an arbitrary number of sockets (but must contain at least one socket). In this case type and name have to be specified for each socket (Figure 7).



The screenshot shows the 'General' tab of the CRT tool. The 'Title' field contains 'G-Prerequisite' and the 'Description' field contains 'GRAPPLE Default: Prerequisite CRT'. The 'Colour / Shape' section has a dropdown set to 'diamond' and a black color swatch. The 'Structure' section has radio buttons for 'Directed' (selected) and 'Undirected'. Below this, two socket configuration boxes are shown, connected by a double line. The left box is for a 'Source Socket' with name 'source', min cardinality '1', and max cardinality '*'. The right box is for a 'Target Socket' with name 'target', min cardinality '1', and max cardinality '*'.

Figure 6: CRT tool: Defining the visual representation and the structure.

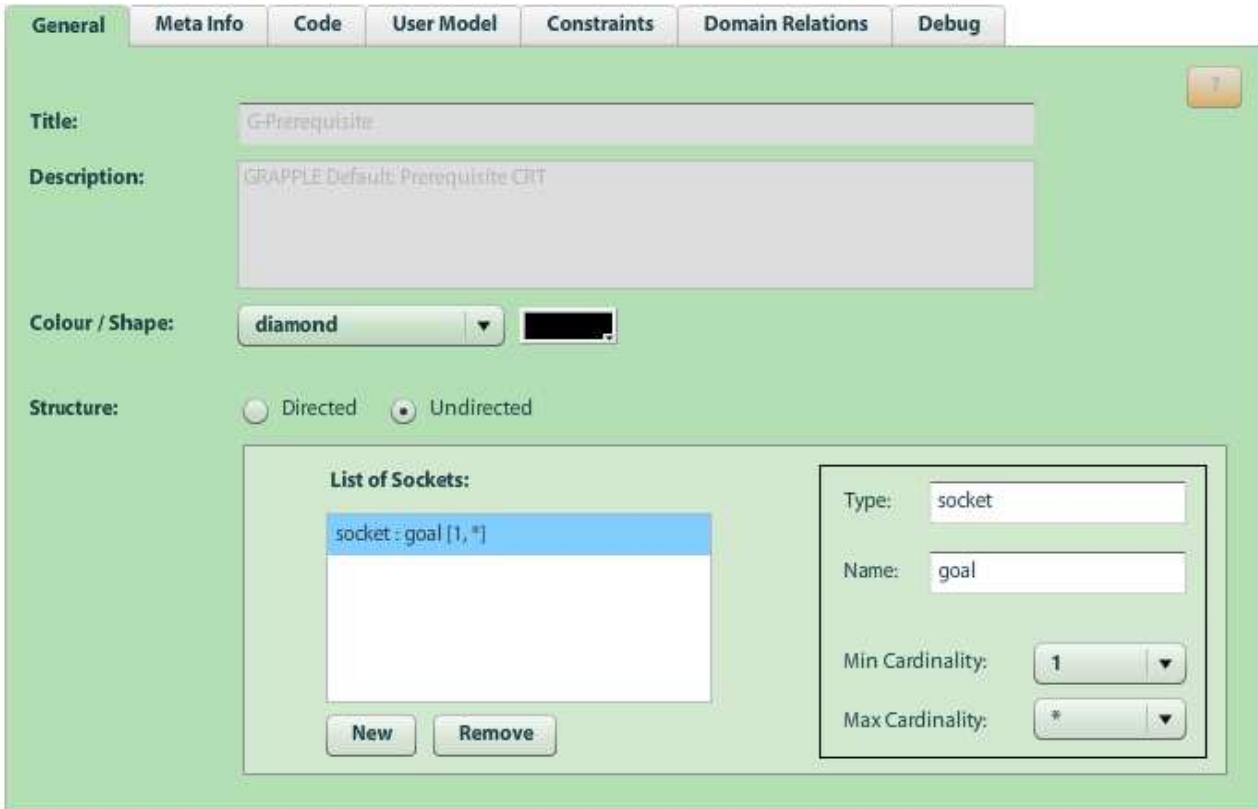


Figure 7: CRT tool: Defining a CRT with an undirected structure and one single socket.

Meta Info Tab

Describing the pedagogical meaning

At the top of the "Meta Info" tab (Figure 8) the pedagogical meaning should be described. This is important information for the author who might choose this CRT in the CAM. On the other hand, this information is free text and will not be interpreted by GALE or any other component.

Additional Information

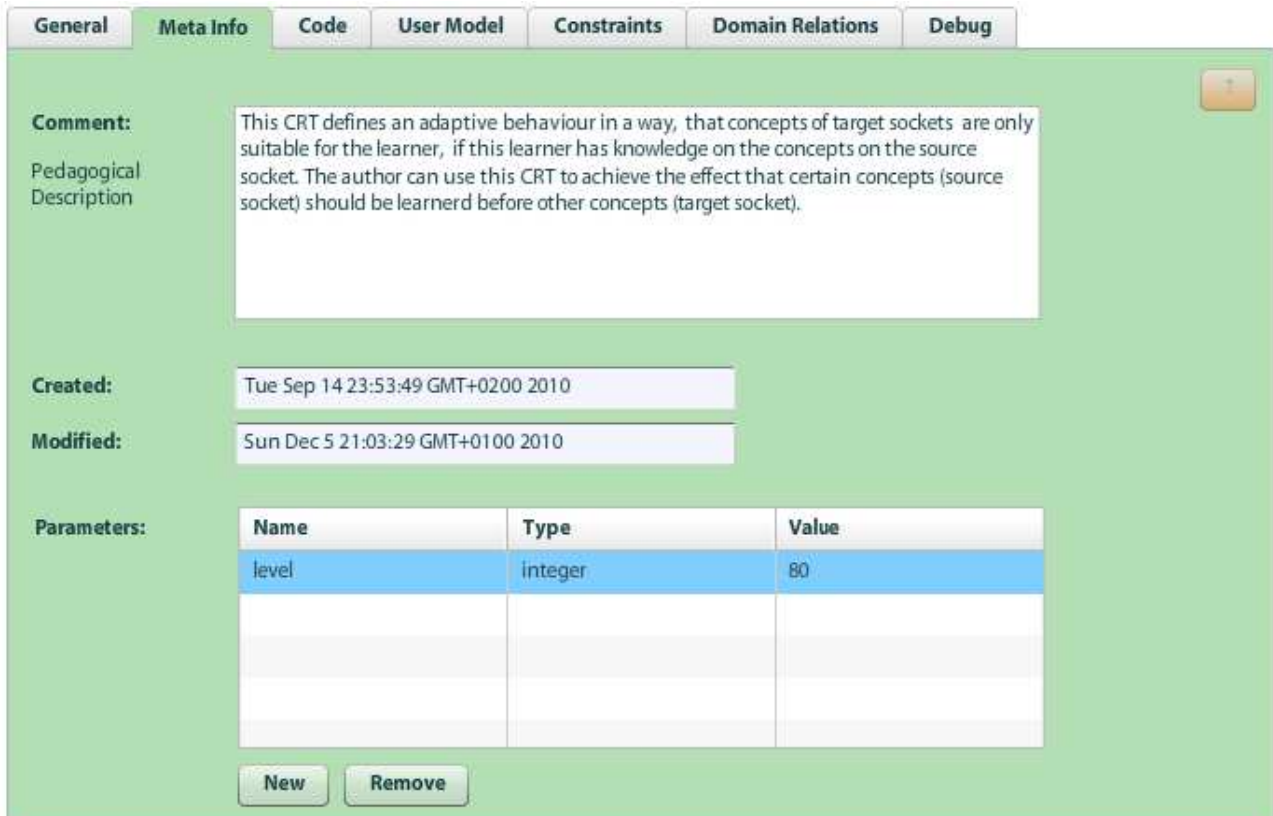
Information on the creation time, modification time and author is also provided (Figure 8). The information of the author is given as UUID, not as a user name. These fields cannot be modified.

Defining parameters in the CRT

The section on the parameters (Figure 8) allows for defining parameters for this CRT. A parameter is a variable an author can assign a value to. The advantage of such a variable is that the value can be assigned in the CAM for certain instances of the CRT. The parameter can be used in the code section (see below) but does not have to be assigned a value at the phase of CRT definition. In the CAM each CRT instance can be assigned a different parameter value, which is then used in GALE after the code is processed.

The example in Figure 8 shows a definition of a 'level' parameter with value 80. This value is the knowledge level which has to be achieved and will make the concepts in the target socket become suitable. If an author uses this CRT in the CAM, he or she can change this level for each single CRT.

An arbitrary number of parameters can be defined. The New and Remove buttons allow for adding new parameters and removing existing parameters. In order to edit a parameter, one only has to click on the respective field in the table and the field can be modified.



General **Meta Info** Code User Model Constraints Domain Relations Debug

Comment:
Pedagogical Description

This CRT defines an adaptive behaviour in a way, that concepts of target sockets are only suitable for the learner, if this learner has knowledge on the concepts on the source socket. The author can use this CRT to achieve the effect that certain concepts (source socket) should be learned before other concepts (target socket).

Created: Tue Sep 14 23:53:49 GMT+0200 2010

Modified: Sun Dec 5 21:03:29 GMT+0100 2010

Parameters:

Name	Type	Value
level	integer	80

New Remove

Figure 8: CRT tool: Pedagogical meaning and parameters of the CRT.

Code Tab

Creating the GALE code

The central part of each CRT is the code that describes the adaptive behaviour of the CRT (see Figure 9). This is the formal representation of the pedagogical meaning. This code fragment has to be written in a language that can be interpreted by GALE. The syntax is described in deliverable D1.1b and is considered a prerequisite to this guide. Theoretically, all other programming languages can be used that GALE is able to process. In this case the type (on top of the text area) has to be set accordingly. However, at the moment only the type "GALE" is provided and can be used.

Even if it is not considered the purpose of this guide to explain GALE code language, two constructs are pointed out here: sockets and user model variables. User model variables are formally defined in a way similar to variables in other programming languages (see below). In the code they can be referred to in sockets meaning that they are applied to all concepts which will be inserted when the CRT is instantiated in the CAM. Later in GALE, when the code is processed, the user model variables are applied to the concepts in the sockets.

The example in the screenshot below (Figure 9) defines a prerequisite relation with the following code:

```
%target%
{
  #suitability & !`(${%source%#knowledge}>%level%)`
}
```

The meaning of this piece of code is that the concepts in the target socket become suitable if the knowledge level of all concepts in the source socket is higher than the value specified in the "level" parameter. The sockets are addressed by their names (see above) using the expression "%socket-name%". The "suitability" variable refers to the "target" socket as it occurs between the brackets of the "%target%" expression. The "knowledge" variable is applied to the concepts in the "source" socket by appending it to the %source%-string. This way user model variables can be connected with sockets.

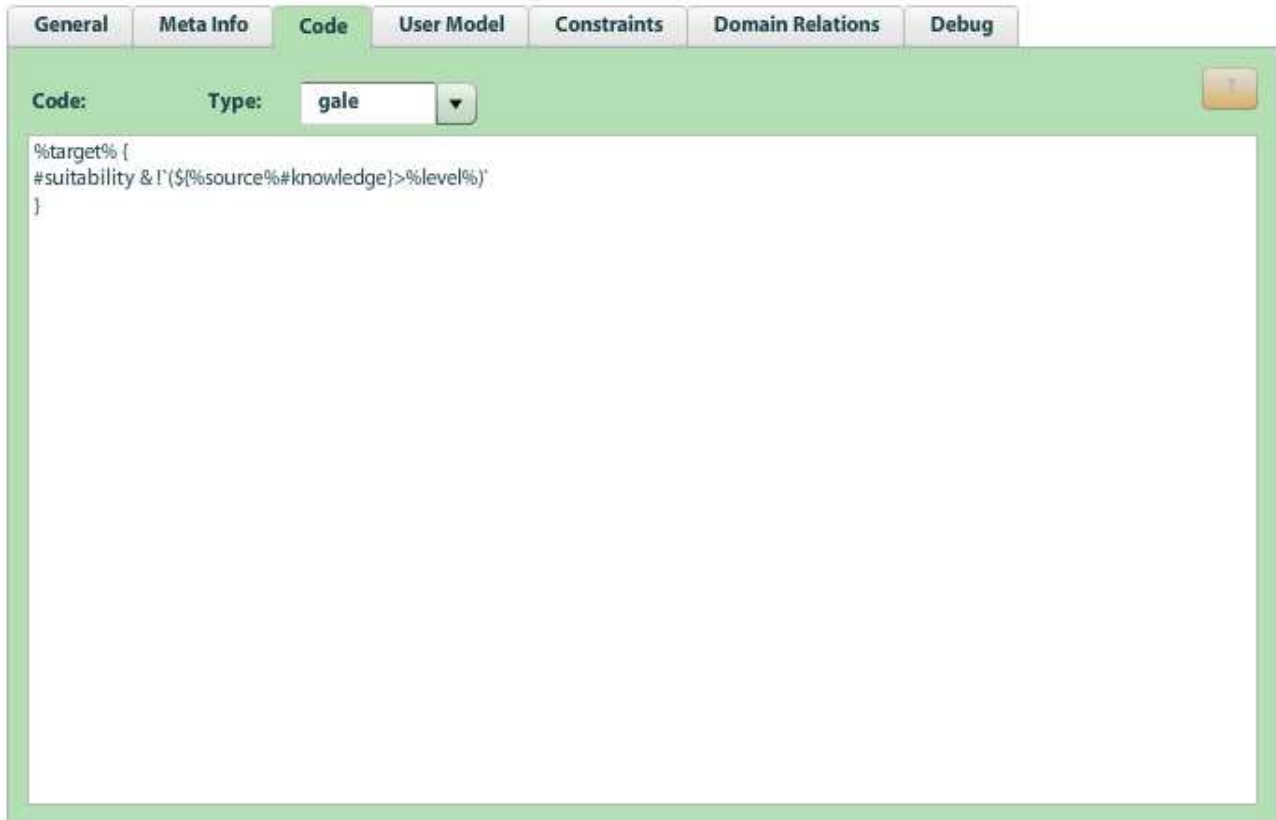


Figure 9: CRT tool: Defining the code of the CRT.

User Model Tab

Defining user model variables in a CRT

The user model variables used in the code have to be formally specified regarding variable type, value range and default value (see Figure 10). The type definition is obligatory, whereas the value range and default value are optional. Another obligatory assignment is the socket to which a user model variable is related. The list of the defined user model variables can be seen in the list box on the left hand side, where user model variable names and assigned sockets are displayed.

It is also possible to specify the "public" and "persistence" state for each variable. A public variable means that it will be shared with GUMF, and that it can be used by other GRAPPLE components, like for example the visualisation tools and widgets. If a variable is not public, it can only be used for the internal adaptive behaviour. A persistent variable means that it will be stored in GALE for further usage. If a variable is not persistent, the adaptive engine has to retrieve its value from GUMF or calculate it if possible.

The location fields of user model variables are not used at the moment.

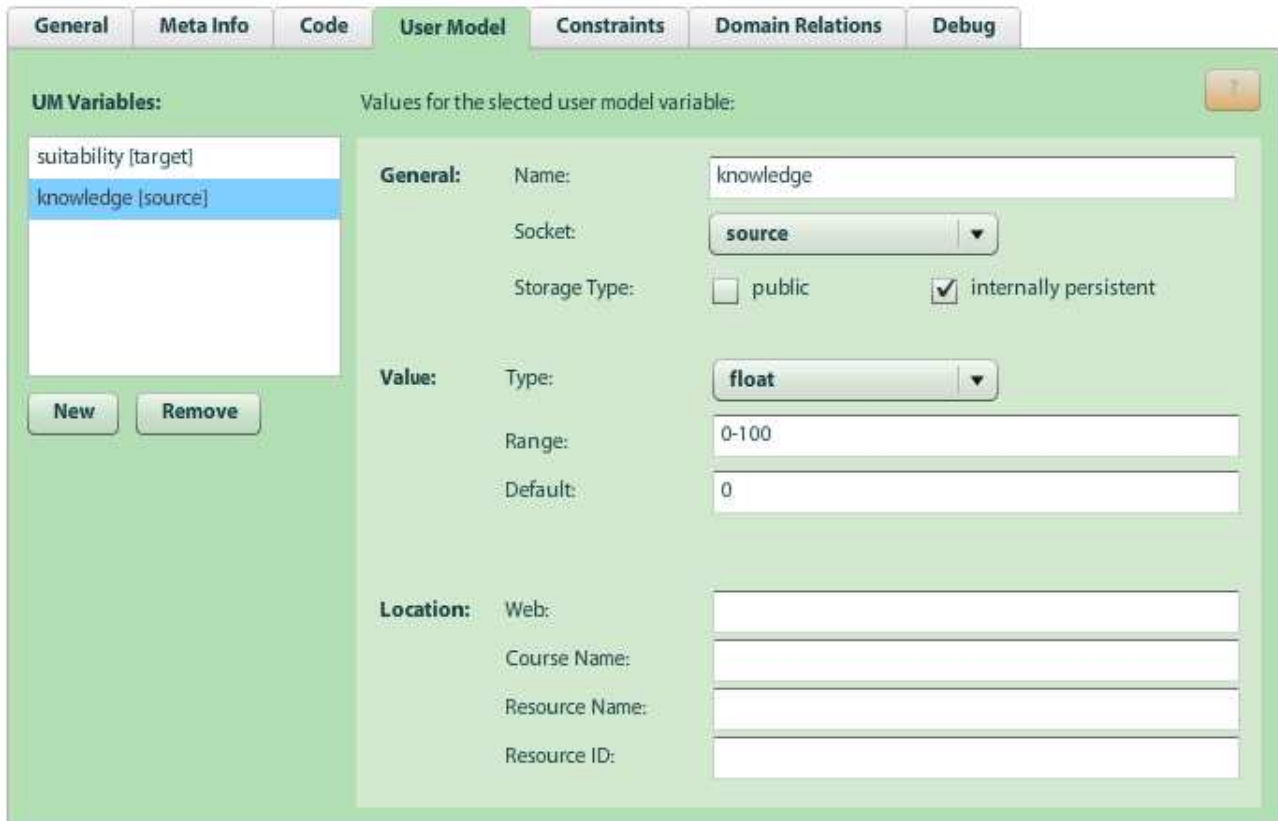


Figure 10: CRT tool: Defining the user model variables of the CRT.

Constraints Tab

Defining Constraints

Theoretically it is possible to define some constraints about how CRTs can be used in the CAM tool (see Figure 11). It is possible to specify if loops are allowed or not, which concepts can be inserted into a socket and constraints regarding neighbourhood of CRTs can be defined. However, at the moment these features are not used in the CAM tool and therefore this specification in the CRT has no effect just yet. Nevertheless, the constraints information is contained in the XML representation of the CRT and stored on in the GAT database together with the CRT.

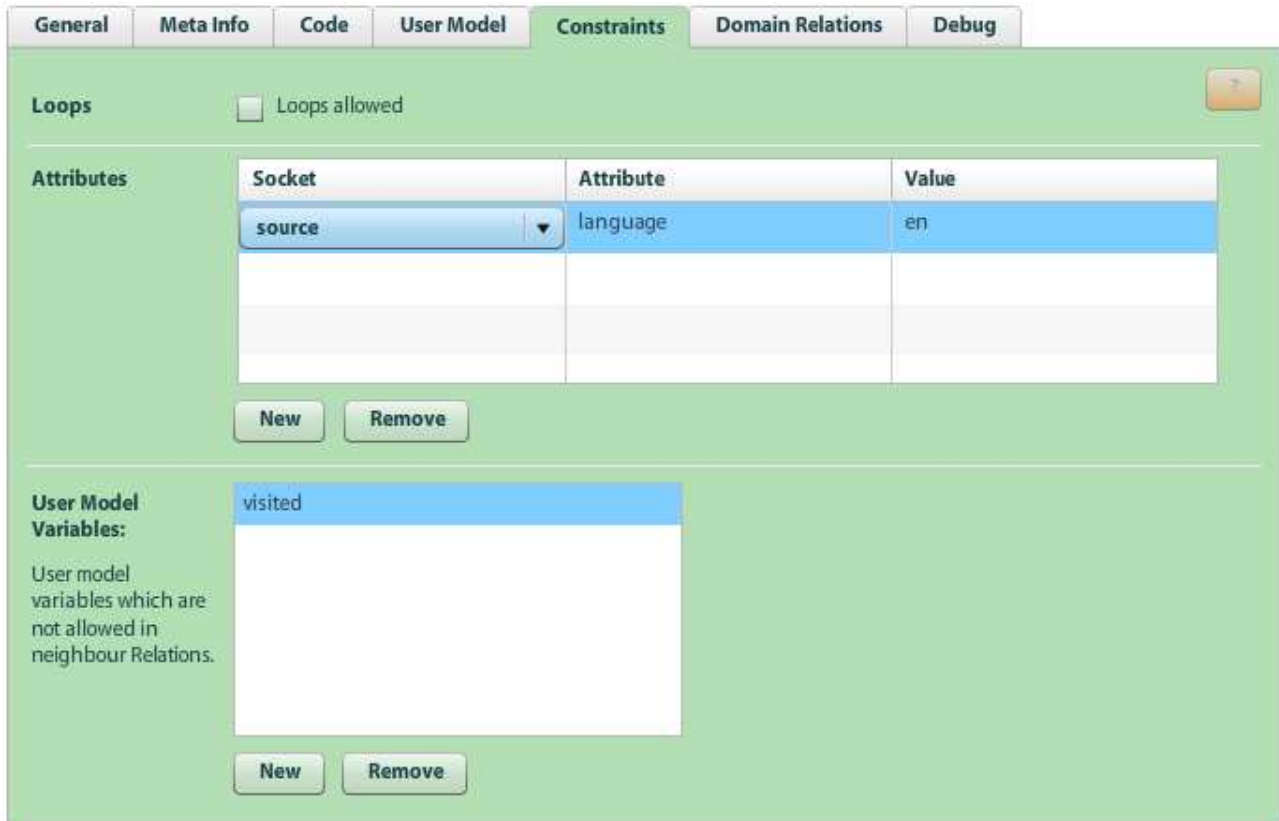


Figure 11: CRT tool: Defining the constraints of the CRT.

Domain Relations Tab

Exploiting Relations of the Domain Model

An additional feature which has not been implemented in the CAM tool yet, is the exploitation of relations between concepts in the domain model. Theoretically, a CRT can be associated with DM relations, which means that the CRT could automatically be created if the specified DM relation is detected in the CAM tool (Figure 12). At the moment this specification has no effect yet. Nevertheless, the domain relations information is contained in the XML representation of the CRT and stored on in the GAT database together with the CRT.

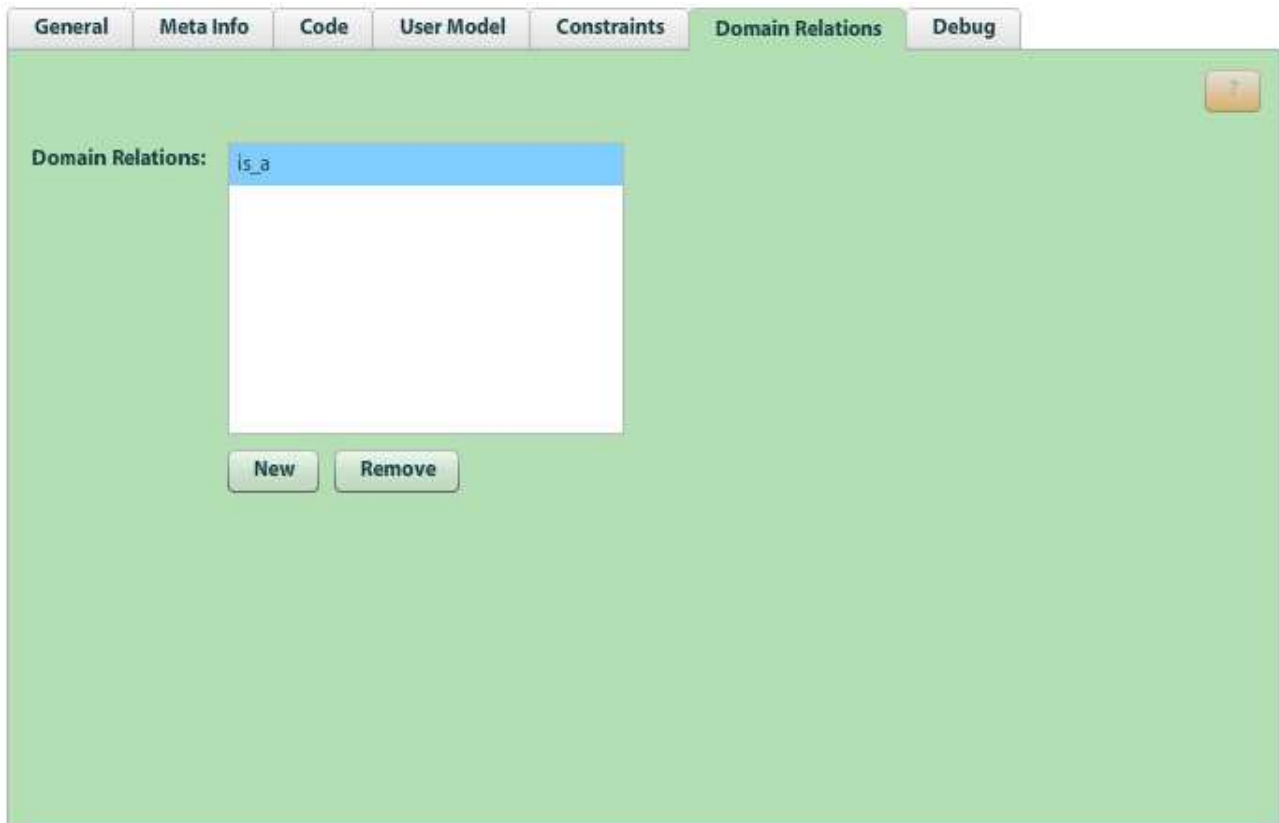


Figure 12: CRT tool: Defining the usage of domain model relations for this CRT.

Debug Tab

XML representation of the current CRT

The XML representation area (Figure 13) is a useful feature for debugging in case there are any problems. By clicking on the "Debug" tab, the CRT is shown in XML representation (as it is stored on the backend). All the graphical specifications described above can be found in the XML code.

It is also possible to modify the XML code and save it back to the tool by clicking on the "Save to Pedagogical Relation Tool" button. The graphical specifications will then be updated according to the modified XML representation.

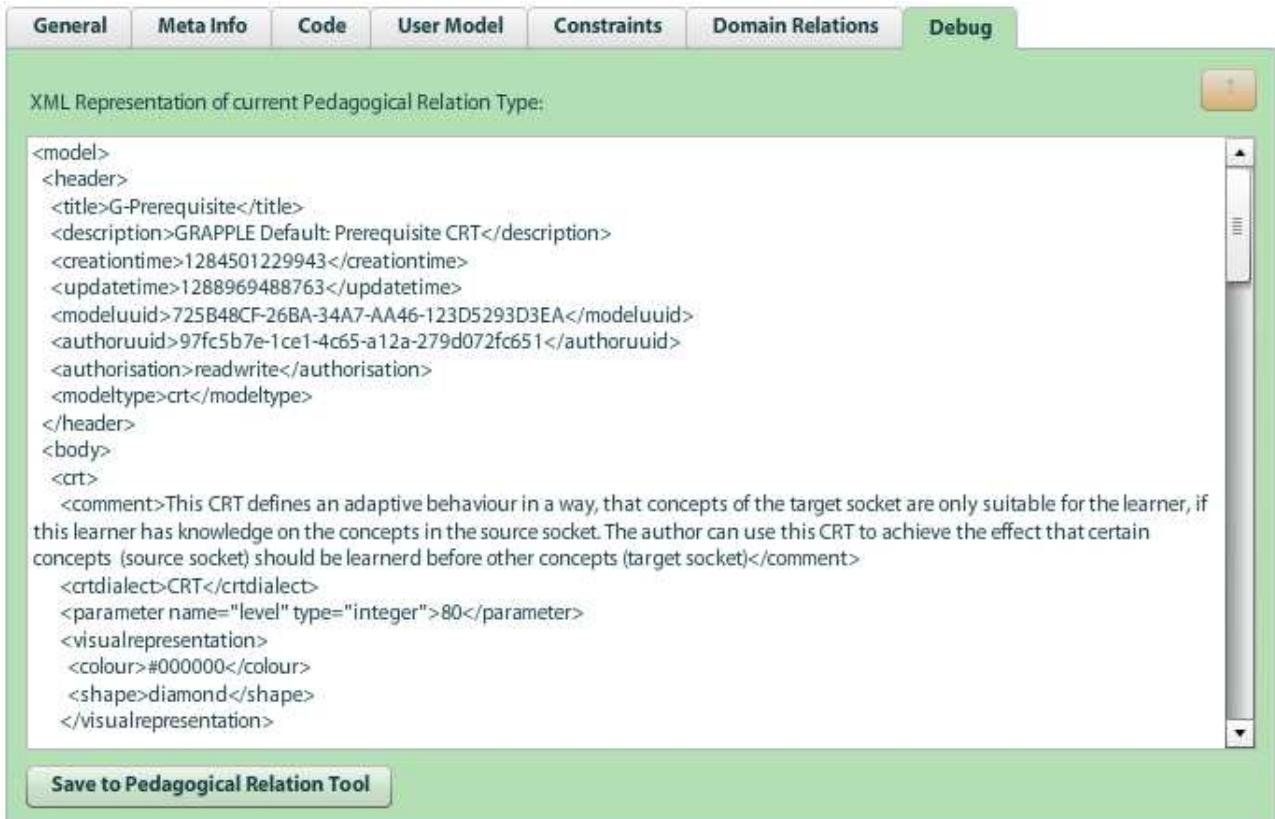


Figure 13: CRT tool: Insight to the XML representation of the current CRT.

5.2 Some Important CRTs as Examples

This section provides a short explanation of four important CRTs, which should also demonstrate the flexibility of CRT definitions. These CRTs are listed in the appendix in their respective XML representation. They are also part of the official GRAPPLE CRTs (shown in the CAM tool by default). An example of the usage of these CRTs is given in deliverable D9.5 in the section on the case study conducted by UniGraz. In this example, concepts are defined in a DM after which a CAM is created where these CRTs and the defined concepts are inserted.

Start CRT

The "Start CRT" is a CRT with an undirected structure and one socket. The purpose of this CRT is to indicate to GALE that a course should start with the concept inserted in this socket. GALE regards every concept in the Start CRT as a separate course, so multiple concepts in this CRT affect multiple courses.

Prerequisite CRT

The "Prerequisite CRT" makes a connection between concepts in terms of which concepts should be presented before other concepts. More precisely, this CRT defines an adaptive behaviour in a way that concepts of the target socket are only suitable for the learner, if this learner has knowledge of the concepts in the source socket. The author can use this CRT to make sure that certain concepts (source socket) are learnt before other concepts (target socket).

Knowledge-Update CRT

The "Knowledge-Update CRT" has an important functionality as it increases the learner's knowledge level of all visited concepts (meaning visited through related learning resources). This is necessary for providing an adaptive behaviour since this information is for example used by the "Prerequisite CRT".

Layout CRT

The "Layout CRT" is used to define the layout of the adaptive course. It can be used to make a layout design that will show all concepts (inserted in this CRT) on a navigation bar (as it is done for the Layout CRT given

in the appendix). In addition, it is important to know that all concepts have to be related to a "parent" relation in the domain model, because GALE needs this information to build the menu hierarchy.

6 GUMF Mapping Tool

This section describes the purpose, design and specification of the GUMF Mapping Tool and provides a user guide for its usage. Please note that this is a new tool and that it has not been described in previous deliverables.

6.1 Purpose of the GUMF Mapping Tool

There was a need for this tool since there was a gap between external variables used in Learning Management Systems (LMS) and user model variables defined in the CRT tool and used in GALE. This gap arises when an author creates a course with the GAT environment and deploys this course to GALE. Next the author creates a test or quiz on the LMS and also captures its result in user model variables stored in GUMF. Even if the quiz is related to the knowledge conveyed in the course, there is no formal relation between these variables and the adaptive engine will not adapt the course to the quiz result.

In order to overcome this situation, the external variables of the quiz have to be related to the user model variables used by GALE. In GUMF the functionality of mapping user model variables has been developed for this purpose. Hence, it is now possible to create rules to achieve this kind of mapping. However, the creation and modification of such rules has to be done using a web interface provided by GUMF. This seems to be difficult for authors, since the interface has not been designed for end-users and authoring takes place somewhere else.

The solution to this problem is a tool integrated in the GAT shell which allows mapping user model variables. It connects to the GUMF and makes use of its mapping functionality. The tool can retrieve existing mapping rules, modify mapping rules in a graphical user interface and store those rules in GUMF again. Since it is integrated with the GAT, the authoring process happens at the same place as for DMs, CRTs, and CAMs.

6.2 Design and Specification of the GUMF Mapping Tool

Mapping rule

From a conceptual point of view a rule (as it is used in the GUMF Mapping Tool) is mapping one user model variable to another user model variable: UM Variable 1 => UM Variable 2. According to the specification of user model variables in GUMF, a user model variable consists of the fields: user, UM variable name, value and level. They are stored as GRAPPLE statements such as 'gc:subject' (the user), 'gc:predicate' (the user model variable name), 'gc:object' (the value or concept), and 'gc:level' (the level). The mapping is modelled as premise - consequent structure (see below).

For example, a user model variable is defined with name "knowledge" and has a range from 0 to 100. A concept "Sun" is inserted in the respective CRT. A quiz has also been created on a LMS, where knowledge is defined through "test-completed" on the test item "test-sun" with a range from 0 to 10 points. A mapping is needed which maps the 'test-completed' to 'knowledge', the test item 'test-sun' to the value/concept 'Sun' and multiplies the level by 10. On a conceptual level the rule would look like this:

```
Rule: [user, test-completed, test item test-sun, level] => [user, knowledge, Sun, multiply by 10]
```

If the learner does the test and achieves a level of 5 points, the user model variable 'knowledge' on the concept 'Sun' is updated and the level is set to 50.

In general, a rule can be expressed like (on a conceptual level):

```
Rule := [user, UM variable name, value, level ] => [ user, UM variable name, value, level ]
```

In order to store and retrieve such a rule, it also has a title and description.

Tool functionality

To provide a tool where such rules can be specified, stored and retrieved, the following functionalities are required:

- 1.) List rules: The author should get a list of available rules in GUMF where it is possible to select a specific rule in order to see its details. GUMF has to be called to receive all mapping rules that have already been created. The name of each of the returned rules can be displayed as a list of "existing mapping rules".
- 2.) Add New Rule: The author should have the possibility to create new rules in a graphical user interface. In order to create a new mapping rule, the rule has to be posted to GUMF in XML format (see below). GAT needs to provide a form where users can specify (at least simple) rules.
- 3.) Remove Rule: Authors should be able to delete single rules which are on GUMF. In order to remove rules, the tool simply has to tell GUMF the ID of the rule that should be deleted and removed.
- 4.) Modifying rules: Authors should have the possibility to modify existing rules. The tools have to be able to retrieve an existing rule from GUMF, provide it to the author in a graphical user interface for modification and send the modified rule back to GUMF. The rule that will be posted to GUMF already has an ID so GUMF knows which rule it has to replace.

Rule format

GUMF expects that rules are formatted in the so-called GRAPPLE Derivation Rule (GDR) language which is a rule language for GRAPPLE statements (GRAPPLE statements are user data information fragments stored in a user profile format as defined in deliverable D2.1).

The rules that should be created with the GAT editor are simple "premise -> consequence" rules and should have the format outlined in the following example:

```
<gdr:rule
  xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/"
  xmlns:gc="http://www.grapple-project.org/grapple-core/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  id="1"
  name="Quiz Level to External Knowledge (CLIX)"
  description="Maps the quiz/test result of a specific score to external knowledge"
  creator="http://pcwin530.win.tue.nl:8080/grapple-umf/client/1">
  <gdr:premise dataspace="1">
    <gc:subject?user</gc:subject>
    <gc:predicate rdf:resource="http://www.grapple-project.org/ims-lip/completedTest"/>
    <gc:object>solar-system-quiz-id</gc:object>
    <gc:level?level</gc:level>
  </gdr:premise>
  <gdr:consequent dataspace="1">
    <gc:subject?user</gc:subject>
    <gc:predicate rdf:resource="http://gale.tue.nl/predicate/knowledge"/>
    <gc:object>gale://gale.tue.nl/cam/DavidTestCAM/SolarSystem</gc:object>
    <gc:level>op:multiply(?level,10)</gc:level>
  </gdr:consequent>
</gdr:rule>
```

This example is basically the same as the example at the beginning of the section. However, it is written in the GDR format.

Tool architecture and implementation

The GAT editor enables the author to deal with mapping rules expressed in XML as outlined above. It has to be able to retrieve a set of rules from GUMF, list them graphically, allow the author to edit them or to create new ones and it has to be able to save them back to the GUMF or delete it from GUMF.

The GUMF Mapping Tool provides a graphical user interface where an author can interact with these rules. They can be listed, modified, created and deleted by the author. The various fields appear as shown in the user guide below (see Figure 17).

The GUMF provides both a RESTful API and an interface through the GRAPPLE Event Bus (GEB). More information on the interface and framework of GUMF can be found in deliverable D6.1b, The functionalities listed above (list rules, create rule, modify rule, delete rule) are exposed on both interfaces. The GUMF Mapping Tool can access the GUMF on both interfaces.

In order to access GUMF and to retrieve, add, delete or modify rules, a dataspace parameter number and a secret token is needed. The dataspace number and token are retrieved when a new dataspace is created.

Currently there is only one dataspace needed in GRAPPLE, where all the user model data are stored. Dataspace number and token are provided as parameter either through the REST interface or the GEB interface. There is however no user specific access rights management that protects rules created by individual authors. Every author has full access to all rules, since dataspace number and token is provided by GAT.

6.3 User Guide to the GUMF Mapping Tool

This section contains a user guide on how to use the GUMF Mapping Tool. According to the functionalities described above, rules can be listed, created, modified and deleted.

In order to open the GUMF Mapping Tool, the author has to start the GAT shell and choose the menu item "Tool / User Model" (see Figure 14). The tool is then opened by disabling the other parts of the GAT shell (see Figure 15) and will appear as a model dialogue.

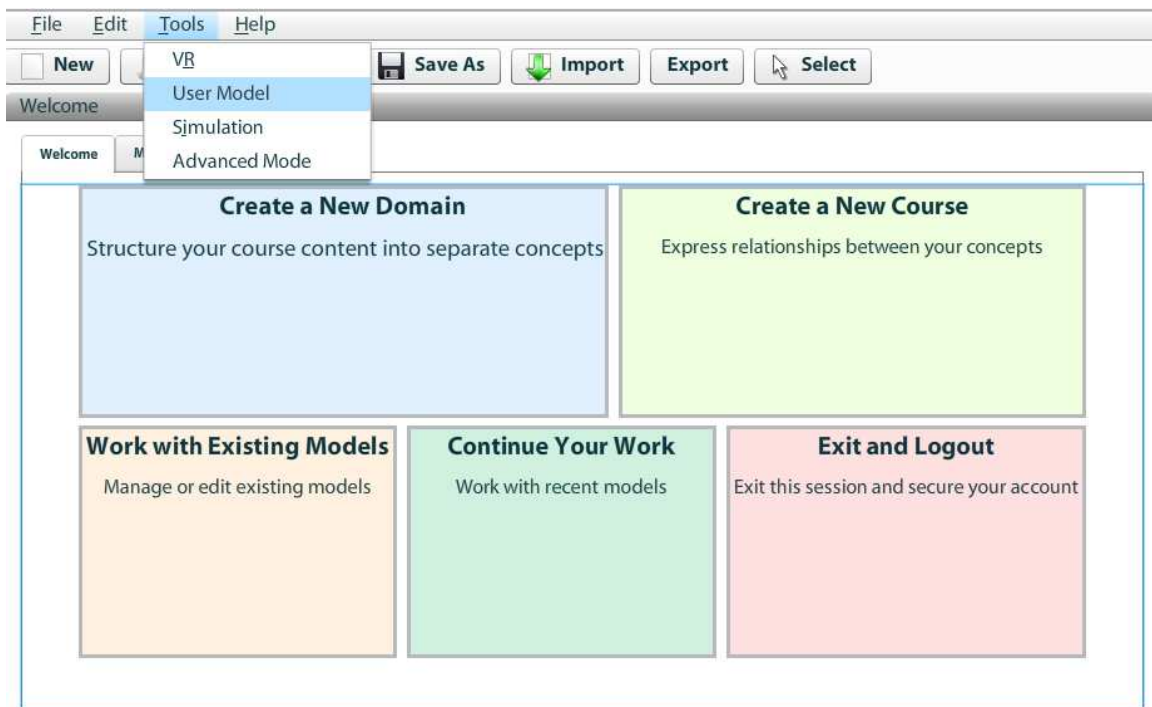


Figure 14: GUMF Mapping Tool: Launching the tool.

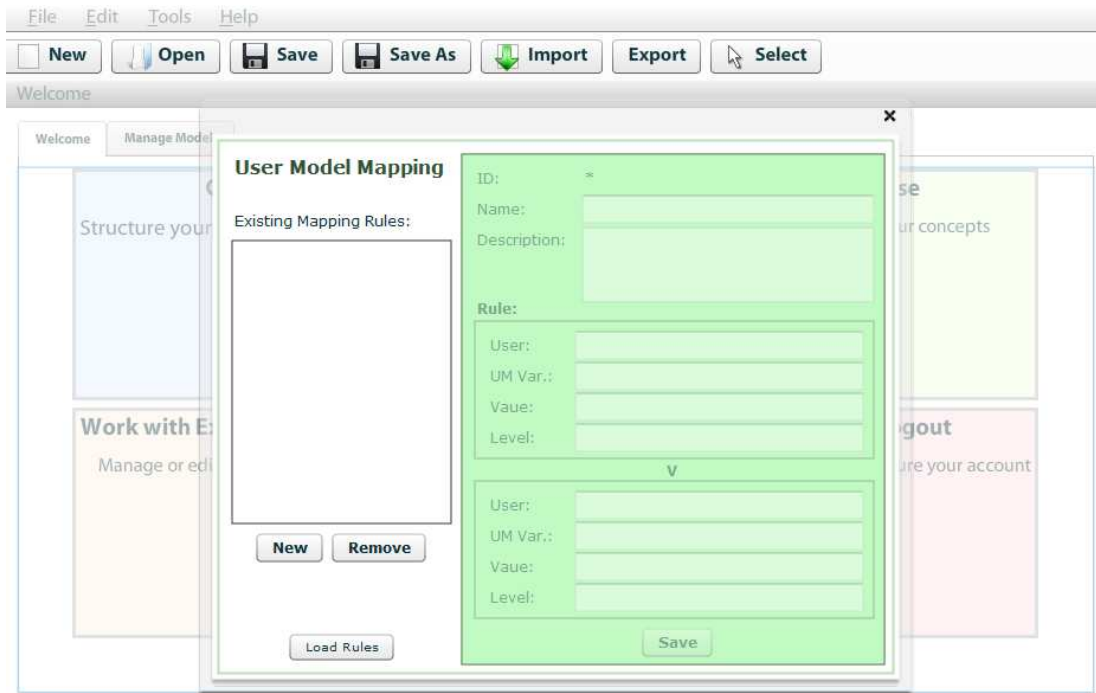


Figure 15: GUMF Mapping Tool: The tool before initialisation.

When the tool appears, it does not display any data. In order to retrieve existing rules, the 'Load Rules' button has to be pressed, which will load all existing rules from GUMF (see Figure 16). As long as no rule is selected the fields for the rules are disabled.

New rules can be created by pressing the 'New' button, which enables the data fields. The author can then specify the title, description and user model variable mapping. Modifying a rule is done in a similar way: By selecting an existing rule, the rule content appears in the data fields which can then be modified. The 'Save' button triggers the storing mechanism to save the selected rule back to GUMF. See Figure 17 regarding enabled data fields.

Deleting a rule is done by selecting the rule and pressing the 'Remove' button. This will delete the rule from the list and also from GUMF.

The meaning of the several data fields (Figure 17) are as follows:

- Name: The name specifies the title of the rule. This is necessary to find existing rules in the list
- Description: A free text description can be entered, which should help authors to understand rules
- Mapping: Two boxes with four fields each appear below the name and description. The box at the top expresses the "source" user model variable mapped to the user model variable expressed in the box at the bottom (destination).

Several fields of both boxes are the same, but depend on their context (e.g. LMS). The author has to identify how a user model variable is represented in the respective context.

- The user field allows for transforming the user information. The expression '?user' represents the current user which can be transformed to the same user by using the same '?user' expression.
- The "UM Var" field refers to the user model variable name. Thanks to the mapping, the two names of the field pair are related to each other.
- The "Value" field refers to the object of the user model variable (in GALE this is the concept). For the mapping the two values/objects are related to each other.
- The "Level" field refers to the level of the user model variable. Mapping the level means that it is transformed from one to the other according to the operation given in the target field. For example, the level can be multiplied by 10 which would be expressed through "op:multiply(?level,10)" using the level from the source field.

For more information on defining the fields, please see deliverable D2.1.

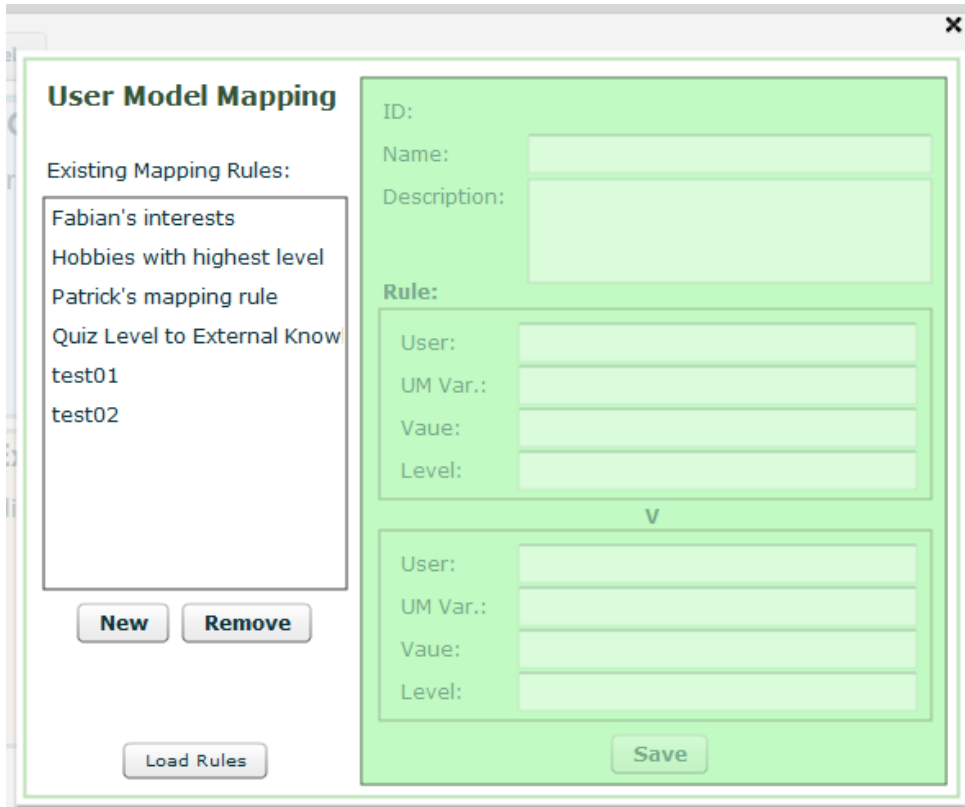


Figure 16: GUMF Mapping Tool: Existing rules have been loaded from GUMF.

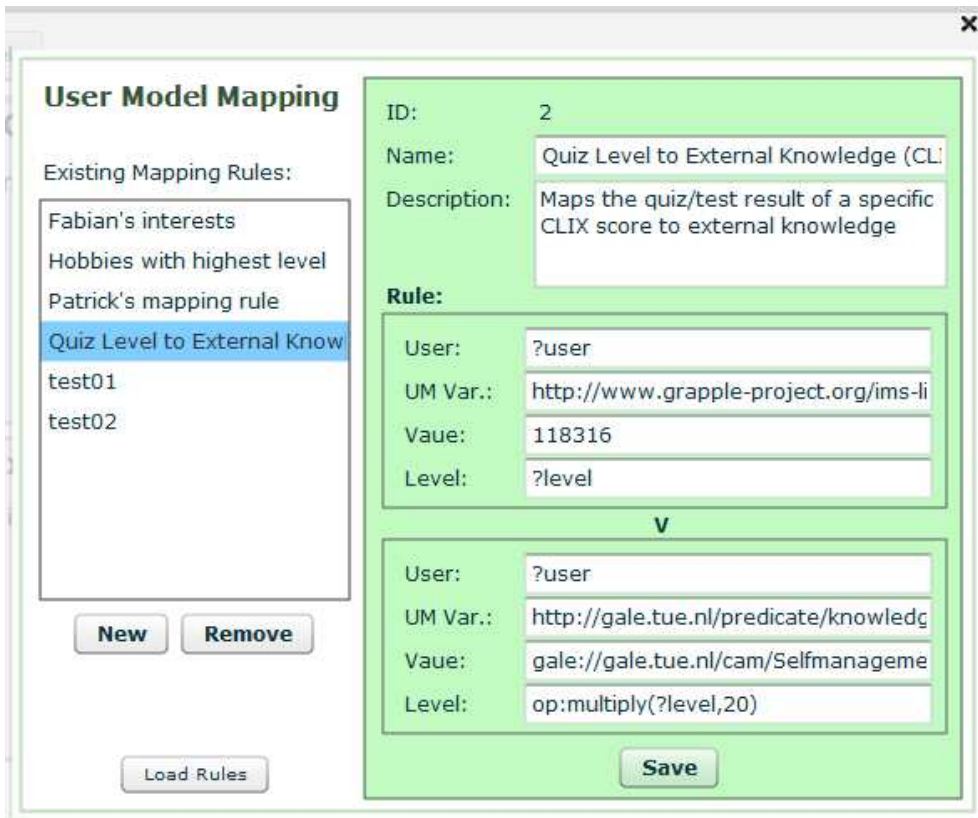


Figure 17: GUMF Mapping Tool: A selected rule.

7 Conclusion

This report presented a description of the final development of the CRT tool as it has been designed in the former deliverable D3.2a and initially described in D3.2b. It shows that this tool is working and fulfils the basic requirements identified in D3.2a. CRTs can be defined and modified and these CRTs can be loaded and saved in XML format to and from a database on the web. The tool is web-based and provides a graphical interface for the author. The CRTs can be used in the CAM tool to create an adaptive course, which can be deployed to GALE.

The initial version of the CRT tool has been demonstrated at the EC-TEL 2009 [3] conference and has been presented in a poster session at the ICCE 2009 [2] conference.

References

1. Adobe Flex (2010). <http://www.adobe.com/products/flex/>, retrieved on 30 July 2010
2. Albert, D., Nussbaumer, A., Steiner, C. M., Hendrix, M., Cristea, A.: Design and Development of an Authoring Tool for Pedagogical Relationship Types between Concepts. Poster at the 17th International Conference on Computers in Education (ICCE 2009), 30 November - 4 December 2009, Hong Kong.
3. Hendrix, M., Nussbaumer, A., Dicerto, M., Oneto, L., Cristea, A.: GAT: The FP7 GRAPPLE Authoring Tool Set. Demonstration at the Fourth European Conference on Technology Enhanced Learning (EC-TEL 2009), 29 September - 2 October 2009, Nice, France.

8 Appendix

8.1 CRT Examples

This appendix section contains the XML representation of the CRTs listed in section 5.2.

Start CRT

```
<model>
  <header>
    <title>G-Start</title>
    <description>GRAPPLE Default: Start CRT</description>
    <creationtime>1286193884782</creationtime>
    <updatetime>1286529210070</updatetime>
    <modeluuid>382B63EE-6467-7AC2-404F-77212F3F8BA6</modeluuid>
    <authoruuid>97fc5b7e-1ce1-4c65-a12a-279d072fc651</authoruuid>
    <authorisation>readwrite</authorisation>
    <modeltype>crt</modeltype>
  </header>
  <body>
    <crt>
      <comment/>
      <crtdialect>CRT</crtdialect>
      <parameter name="" type=""/>
      <visualrepresentation>
        <colour>#000000</colour>
        <shape>diamond</shape>
      </visualrepresentation>
      <adaptationbehaviour>
        <code type="gale"><![CDATA[%self% {
start `true`
}]]></code>
      </adaptationbehaviour>
    </crt>
  </body>
</model>
```

```

<constraints>
  <loopsallowed>>true</loopsallowed>
  <attributeconstraints/>
  <umvariableconstraints/>
</constraints>
<crtsockets>
  <socket type="self">
    <uuid>f67ed906-ab49-604a-ad0e-1238199ba589</uuid>
    <name>self</name>
    <mincardinality>1</mincardinality>
    <maxcardinality>*</maxcardinality>
  </socket>
</crtsockets>
<associateddmrelations/>
</crt>
</body>
</model>
    
```

Knowledge-Update CRT

```

<model>
  <header>
    <title>G-Knowledge-Update</title>
    <description>GRAPPLE Default: Knowledge Update CRT</description>
    <creationtime>1284502213193</creationtime>
    <updatetime>1288178730330</updatetime>
    <modeluuid>FF78DE75-B152-2F3D-DAB2-124C537D20A3</modeluuid>
    <authoruuid>97fc5b7e-1ce1-4c65-a12a-279d072fc651</authoruuid>
    <authorisation>readwrite</authorisation>
    <modeltype>crt</modeltype>
  </header>
  <body>
    <crt>
      <comment/>
      <crtdialect>CRT</crtdialect>
      <parameter name="level" type="integer">35</parameter>
      <visualrepresentation>
        <colour>#000000</colour>
        <shape>diamond</shape>
      </visualrepresentation>
      <adaptationbehaviour>
        <code type="gale"><![CDATA[%self% {
          event + `
          if ({#suitability}) {
            ##visited, ${#visited}+1};
            ##own_knowledge, 100};
          }
          if (!${#suitability} && ${#own_knowledge} < %level%)
            ##own_knowledge, %level%;
          `

          #own_knowledge {event + `
          ##knowledge, ${#knowledge}+changed.diff/(${<- (parent)}.length+1)};
          `}

          #knowledge {event + `
          if (${<- (parent)}.length > 0)
            ##->(parent)#knowledge, ${<- (parent)#knowledge}+changed.diff/(${<- (parent)<-
            (parent)}.length+1)};
          `}
        }]]></code>
      <usermodel>
        <umvariable>
          <umvarname>knowledge</umvarname>
          <socket>self</socket>
          <public>>false</public>
          <persistent>>true</persistent>
          <location>
            <web/>
            <remotecourse>
              <remotecoursename/>
              <resource>
                <resourceuniqueid/>
                <resourcename/>
              </resource>
            </remotecourse>
          </location>
          <type>float</type>
        </umvariable>
      </usermodel>
    </crt>
  </body>
</model>
    
```

```

    <range>
      <from>0</from>
      <to>100</to>
    </range>
    <default><![CDATA[0]]></default>
  </umvariable>
  <umvariable>
    <umvarname>own_knowledge</umvarname>
    <socket>self</socket>
    <public>>false</public>
    <persistent>>true</persistent>
    <location>
      <web/>
      <remotecourse>
        <remotecoursename/>
        <resource>
          <resourceuniqueid/>
          <resourcename/>
        </resource>
      </remotecourse>
    </location>
    <type>float</type>
    <range>
      <from>0</from>
      <to>100</to>
    </range>
    <default><![CDATA[0]]></default>
  </umvariable>
  <umvariable>
    <umvarname>visited</umvarname>
    <socket>self</socket>
    <public>>false</public>
    <persistent>>true</persistent>
    <location>
      <web/>
      <remotecourse>
        <remotecoursename/>
        <resource>
          <resourceuniqueid/>
          <resourcename/>
        </resource>
      </remotecourse>
    </location>
    <type>integer</type>
    <range/>
    <default><![CDATA[0]]></default>
  </umvariable>
</usermodel>
</adaptationbehaviour>
<constraints>
  <loopsallowed>>false</loopsallowed>
  <attributeconstraints/>
  <umvariableconstraints/>
</constraints>
<crtsockets>
  <socket type="self">
    <uuid>e44eb8bd-ed64-8122-b9b7-1251f34cd392</uuid>
    <name>self</name>
    <mincardinality>1</mincardinality>
    <maxcardinality>*</maxcardinality>
  </socket>
</crtsockets>
<associateddmrelations/>
</crt>
</body>
</model>

```

Prerequisite CRT

```

<model>
  <header>
    <title>G-Prerequisite</title>
    <description>GRAPPLE Default: Prerequisite CRT</description>
    <creationtime>1284501229943</creationtime>
    <updatetime>1287647271603</updatetime>
    <modeluuid>725B48CF-26BA-34A7-AA46-123D5293D3EA</modeluuid>
    <authoruuid>97fc5b7e-1ce1-4c65-a12a-279d072fc651</authoruuid>
    <authorisation>readwrite</authorisation>
  </header>

```

```

<modeltype>crt</modeltype>
</header>
<body>
  <crt>
    <comment/>
    <crtdialect>CRT</crtdialect>
    <parameter name="level" type="integer">80</parameter>
    <visualrepresentation>
      <colour>#000000</colour>
      <shape>diamond</shape>
    </visualrepresentation>
    <adaptationbehaviour>
      <code type="gale"><![CDATA[%target% {
#suitability & !`(${{source%#knowledge}>%level%}`
}
]]></code>
    <usermodel>
      <umvariable>
        <umvarname>suitability</umvarname>
        <socket>target</socket>
        <public>>false</public>
        <persistent>>false</persistent>
        <location>
          <web/>
          <remotecourse>
            <remotecoursename/>
            <resource>
              <resourceuniqueid/>
              <resourcename/>
            </resource>
          </remotecourse>
        </location>
        <type>boolean</type>
        <range/>
        <default><![CDATA[true]]></default>
      </umvariable>
      <umvariable>
        <umvarname>knowledge</umvarname>
        <socket>source</socket>
        <public>>false</public>
        <persistent>>true</persistent>
        <location>
          <web/>
          <remotecourse>
            <remotecoursename/>
            <resource>
              <resourceuniqueid/>
              <resourcename/>
            </resource>
          </remotecourse>
        </location>
        <type>integer</type>
        <range>
          <from>0</from>
          <to>100</to>
        </range>
        <default><![CDATA[0]]></default>
      </umvariable>
    </usermodel>
  </adaptationbehaviour>
  <constraints>
    <loopsallowed>>false</loopsallowed>
    <attributeconstraints/>
    <umvariableconstraints/>
  </constraints>
  <crtsockets>
    <socket type="source">
      <uuid>ac7024e6-6171-a051-648a-123d5293f4ad</uuid>
      <name>source</name>
      <mincardinality>1</mincardinality>
      <maxcardinality>*</maxcardinality>
    </socket>
    <socket type="target">
      <uuid>373727e1-82b0-8a6b-5b09-123d529304dd</uuid>
      <name>target</name>
      <mincardinality>1</mincardinality>
      <maxcardinality>*</maxcardinality>
    </socket>
  </crtsockets>

```

```

    <associateddmrelations/>
  </crt>
</body>
</model>

```

Layout CRT

```

<model>
  <header>
    <title>G-Layout</title>
    <description>GRAPPLE Default: Layout CRT</description>
    <creationtime>1284503327942</creationtime>
    <updatetime>1288453358637</updatetime>
    <modeluuid>1F0BCF82-0479-0284-937F-125D56177F31</modeluuid>
    <authoruuid>97fc5b7e-1ce1-4c65-a12a-279d072fc651</authoruuid>
    <authorisation>readwrite</authorisation>
    <modeltype>crt</modeltype>
  </header>
  <body>
    <crt>
      <comment/>
      <crtdialect>CRT</crtdialect>
      <parameter name="" type=""/>
      <visualrepresentation>
        <colour>#000000</colour>
        <shape>diamond</shape>
      </visualrepresentation>
      <adaptationbehaviour>
        <code type="gale"><![CDATA[%self% {
#layout:String `
<struct cols="250px;*">
  <view name="static-tree-view" />
  <struct rows="60px;*;40px">
    <view name="file-view" file="gale:/grapple/header.xhtml" />
    <content />
    <p><hr />Next suggested concept to study: <view name="next-view" /></p>
  </struct>
</struct>
`}}]></code>
        </adaptationbehaviour>
      </constraints>
      <constraints>
        <loopsallowed>>false</loopsallowed>
        <attributeconstraints/>
        <umvariableconstraints/>
      </constraints>
      <crtsockets>
        <socket type="self">
          <uuid>05231dc9-b353-d359-57db-125d9835f6e5</uuid>
          <name>self</name>
          <mincardinality>*</mincardinality>
          <maxcardinality>*</maxcardinality>
        </socket>
      </crtsockets>
    </adaptationbehaviour>
  </crt>
</body>
</model>

```