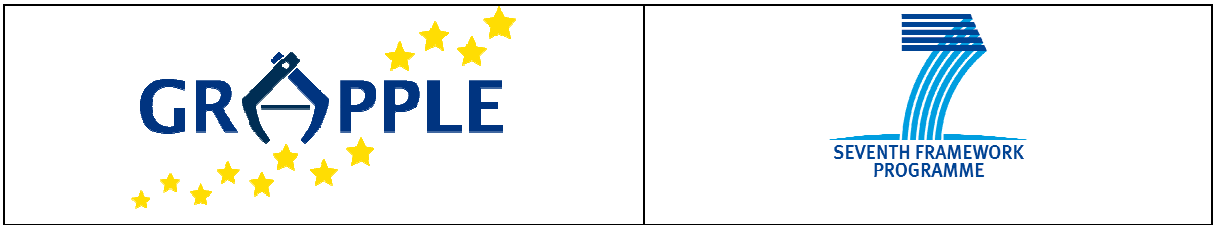

GRAPPLE

3.2b Version: 1.0

Initial Implementation of the Concept Relationship Type Tool

Document Type	Deliverable
Editor(s):	Christina M. Steiner (UniGraz), Alexander Nussbaumer (UniGraz)
Author(s):	Alexander Nussbaumer (UniGraz), Christina M. Steiner (UniGraz), Maurice Hendrix (Warwick), Alexandra I. Cristea (Warwick)
Reviewer(s):	Riccardo Mazza (USI) and Frederic Minne (UCL)
Work Package:	WP3
Due Date:	M18
Version:	1.0
Version Date:	2010-01-25
Total number of pages:	24

Abstract: This report describes the initial 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). Updates of the CRT definition and a technical specification are given in this deliverable. The initial implementation of the CRT Tool is reported which is based on the findings of the previous deliverable and the updates of the CRT definition. Furthermore the integration of the CRT Tool with the other authoring tools is described. Finally the graphical user interface is described, which explains how CRTs can be defined and which should provide help for the author.

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 for defining CRTs. Based on this work the CRT Tool has been implemented. This report documents the work done in the context of the development of the CRT Tool. Report and implementation together form the Deliverable D3.2b.

CRTs have been technically specified using XML format. An XML schema has been defined which 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 of the CRT Tool to the other authoring tools (DM Tool and CAM Tool), a common software library has been created which offers functionalities to all tools. Together the tools and the software library form the Grapple Authoring Toolset (GAT). This report describes how the CRT Tool is included in the GAT shell.

Improvements and planned work for the final implementation of the CRT Tool (D3.2c) are outlined. Advancing the user interface and making it more intuitive for authors is an important plan for improving the tool. Furthermore, the result of the evaluation of the authoring tools will influence the further development.

The CRT Tool will be demonstrated at the EC-TEL 2009 [3] conference and presented in a poster session at the ICCE 2009 [2] conference.

Authors

Person	Email	Partner code
Alexander Nussbaumer	alexander.nussbaumer@uni-graz.at	UniGraz
Christina M. Steiner	chr.steiner@uni-graz.at	UniGraz
Maurice Hendrix	maurice@dcs.warwick.ac.uk	Warwick
Alexandra I. Cristea	acristea@dcs.warwick.ac.uk	Warwick

Table of Contents

Tables and Figures

List of Figures

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 Tool Set
GALE	GRAPPLE Adaptive Learning Environment
GUMF	GRAPPLE User Model Framework
GRAPPLE	Generic Responsive Adaptive Personalized Learning Environment
GUI	Graphical User Interface
UM	User Model

1 Introduction

By using Concept Relationship Types (CRT) a teaching strategy can be created. With CRTs concepts from the domain model (DM) can be connected to form 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.

In order to empower the content author to define CRTs the CRT Tool has been implemented. This report describes the CRT Tool from a technical perspective, which includes the XML format how CRTs are expressed, the software architecture, and the graphical user interface.

In Section 2 the specification of the CRT format is described. 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 in by means of a XML schema. Finally some revisions are listed which have been made since the last deliverable on CRTs.

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 with the GRAPPLE authoring toolset (GAT) shell, the 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 more detailed user guide is given in the Deliverable on training (D9.2).

In the last section (Section 5) the possibilities and plans for improvement are mentioned. Most importantly the user interface should be made more intuitive and the results of the evaluation will influence the further work on the CRT Tool.

2 Specification of the CRT Format

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

CRTs are expressed in XML format and follow a 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 3.2a. Two complete examples of CRT definitions can be found in the Appendix

2.1 Structure of the CRT format

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

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

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

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

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

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

(1) CRT dialect

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

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

(2) Comment

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

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

(3) Visual representation

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

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

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

(4) CRT Sockets

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

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

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

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

```
<crtsockets>
```

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

(5) Adaptation behaviour

The definition of the adaptation behaviour is the central part of the CRT definition. It defines in a machine-readable way the pedagogical meaning of the CRT. The definition of the adaptation behaviour is done through a piece of code written in Grapple Adaptation Language (GAL). The GAL code is inserted into the XML code as it is (using CDATA). The specification of the GAL code (syntax and meaning) is done in WP1. The following example shows how the GAL code is inserted.

```
<galcode>
  <![CDATA[
    // gale code here
  ]]>
</galcode>
```

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

```
<adaptationbehaviour>
  <usermodel>
    <umvariable>
      // specification of the fisrt user model variable
    </umvariable>
    <umvariable>
      // specification of the second user model variable
    </umvariable>
  <usermodel>
  <galcode>
    <![CDATA[
      // gale code here
    ]]>
  </galcode>
<adaptationbehaviour>
```

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

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

```
    <web>http://www.mySakaiInstallation.com/course</web>
    <remotecourse>
      <remotecoursename>myCourseOnSakai</remotecoursename>
        <resource>
          <resourceuniqueid>cf5de7f5-12b5-4720-92e5-ttttttttttttt</resourceuniqueid>
          <resourcename>myTest</resourcename>
        </resource>
     </remotecourse>
  </location>
</umvariable>
```

(6) Constraints

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

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

(7) Associated domain model relations

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

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

2.2 XML Schema

This section contains the XML schema specification of the CRT XML format. CRTs which are 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. Two 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="modeltype" type="modeltypetype" 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="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:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="body">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="crt" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="crtdialect" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="comment" minOccurs="1" 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:sequence>
                </xs:complexType>
              </xs:element>
           </xs:sequence>
          </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 seven elements of the CRT definition -->

  <!--  (B.1) the CRT dialect  -->
  <xs:element name="crtdialect" type="crtdialecttype"/>

  <xs:simpleType name="crtdialecttype">
    <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 visual representation -->
  <!-- colour format: 0xrrggbb -->
  <xs:element name="visualrepresentation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="shape" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="colour" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- (B.4) the sockets -->
  <!-- mincardinality and  mincardinality: either '*' or any integer value >= 0 -->
  <!-- attribute type: either pair of soeckts with type 'source' and 'target'
    or any number of sockets with any type -->
  <xs:element name="crtsockets" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="socket" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
               <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
```

```
            <xs:element name="uuid" 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:element name="colour" type="xs:string" minOccurs="0" maxOccurs="1"/>
          </xs:sequence>
          <xs:attribute name="type" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- (B.5) the adaptation behaviour -->
<!-- galcode element: contains CDATA -->
<!-- um var type: can be any type, default: according to type (string is enough)
  range: depending on the type, so anyType has been chosen, may conatin further xml code -->
 <xs:element name="adaptationbehaviour">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="usermodel" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="umvariable" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="galcode" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="umvariable">
  <xs:complexType>
    <xs:sequence>
       <xs:element name="umvarname" 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" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="web" minOccurs="0" maxOccurs="1"/>
            <xs:element name="remotecourse" type="xs:anyType" minOccurs="0" maxOccurs="1"/>
          </xs:sequence>
        </xs:complexType>
       </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- (B.6) constraints -->
<xs:element name="constraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="allowedinloop" 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 name="umvariableconstraints">
```

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

    <!-- (B.6) associated domain model relations. -->
    <xs:element name="associateddmrelations">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="relationshiptype" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:schema>
```

2.3 Revisions

There is an important change on the CRT definition since the last deliverable on CRTs (D3.2a). The construct of sockets has been introduced which allows for a more general way of expressing how CRTs are related to each other. In the last deliverable, it was explicitly defined how many concepts are related with how many concepts. In the new version this relation is implicitly defined, since concepts are collected in sockets and sockets are related to each other.

Another change on the CRT format is done in terms of user model variables. In the GAL code user model variables are used in order to update them. In this version all used user model variables used in the GAL code are explicitly defined in the CRT.

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 a using Web service and a database behind the Web service.

For the reason of a more efficient implementation the common functionalities of DM Tool, CRT Tool, and 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 container including a menu bar and handling the creation, loading, and saving of data models.

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.

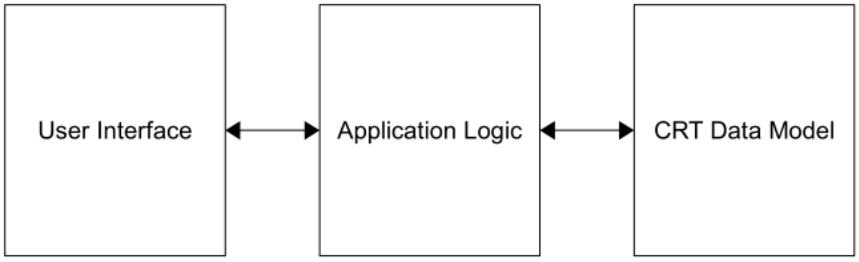

Figure 1: The CRT Tool architecture designed as Model-View-Controller design pattern

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 explanation of the UI are described in Section 4.

The application logic is the component which handles the events from the UI. Since parts of the CRT are related or dependent from each other, modifications on these parts may affect other parts of the CRT. Furthermore, the application logic is responsible for handing 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 XML representation. Since there are several classes representing specific parts of the CRT, each of these classes is responsible for the conversion to and from XML representation.

Implementation of the CRT Tool has been done in Adobe Flex [1] programming language. Flex offers good support for Web-based and graphical applications. The definition of the user interface is represented as 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 installed the Flash plug-in.

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. Figure 2 depicts how the CRT Tool is integrated with the GAT shell.

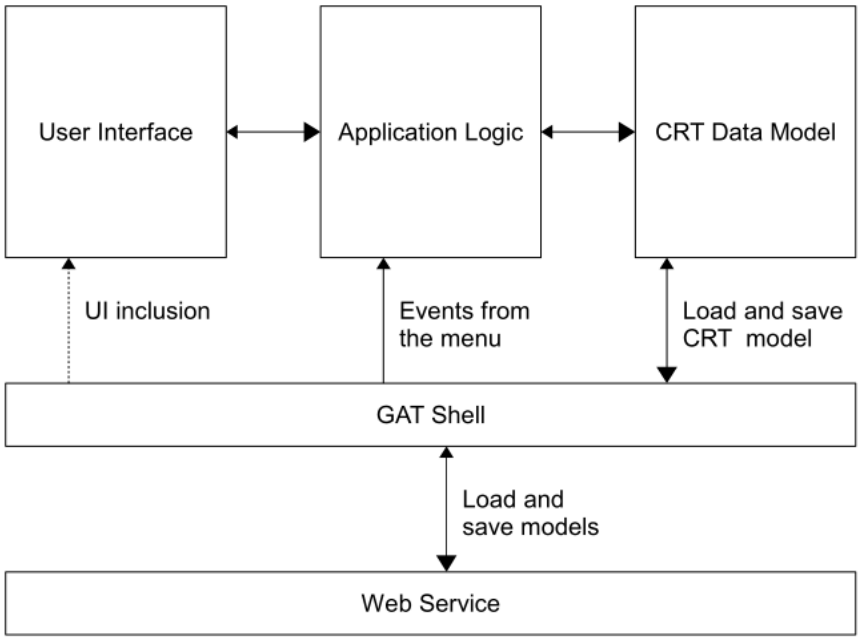

Figure 2: The CRT Tool integrated in the GAT shell

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 transformation of the internal CRT representation to and from XML format. The XML code and be sent to or retrieved from the GAT shell. The GAT shell 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 easily define CRTs. 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 initial implementation is the first

version which will continuously be updated in further work. Anyway, the current version of the GUI provides the possibility to edit all parts of the CRT so that a complete CRT can be defined.

The information needed to specify a CRT includes:

- general information of CRTs, such as name and description, comment, creation time, and author
- information how an instance of a CRT should be visually represented in the CAM
- the structure of the CRT defined by so-called sockets and their properties
- the GAL code which formally defines the adaptation behaviour
- the user variables which the GAL code is accessing
- the properties of the source and target socket which the CRT connects
- 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
- relations to domain model relations

According to these parts of the CRT format specification there are several sections in the GUI in order to define the respective parts in the CRT. These sections are structured with tabs which the author can use to switch between the different parts of information. In the following the tabs are explained using screenshots of each of them.

Tab 1: CRT General

In the first tab ("CRT General", Figure 3), the following CRT properties can be specified:

- the title of the CRT
- the description of the CRT, which should be a general verbal description
- the visual representation consisting of the colour and the shape is specified; this defines how a CRT is represented in the CAM tool.
- 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 name of the socket and minimum and maximum number of concepts which can be placed in a socket. Furthermore there are two structures: directed and undirected.
    - 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.

Tab 2: Meta-Data

In the second tab ("Meta-Info", Figure 4), the following CRT properties can be specified:

- comments include the pedagogical meaning of a CRT. The author can describe in free text the pedagogical meaning, which is not processed by GRAPPLE
- creation and modification time: 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)

Tab 3: GAL Code

In the third tab ("GAL Code", Figure 5), the following CRT properties can be specified:

- GAL code: the GAL code defines the adaptive behaviour (the meaning of the CRT on a technical level). The code has to be input in this text area as it is done for programming languages. This piece of code will be interpreted by the GRAPPLE adaptive engine.

Tab 4: User Model

In the forth tab ("User Model", Figure 6), the user model variables accessed by the GAL code can be specified. UM variables can be added or removed and for each variable the following properties can be specified:

General information:

- name: the name of the user model variable
- socket: the socket can be specified on which this UM variable is applied
- Storage type:
    - public which defines whether GALE needs to submit updates to GUMF or not
    - persistent which defines whether the value is stored or computed or retrieved by GALE

Value information

- the type of the UM variable can be defined which can be one of integer, float, string, enum, or boolean. The type determines which values can be expressed by this user model variable.
- range and default: the range limits the range of values according to the specified type.

Location

- the location determines where the user model variables occur (if it is mapped from an LMS outside of the GRAPPLE engine).

Tab 5: Constraints

In the fifth tab ("Constraints", Figure 7), constraints can be defined which limit the usage of CRTs in the CAM tool. The following constraints can be specified:

- loops: this flag defines if it is allowed to create loops in the CAM tool with this CRT.
- attributes: this table restricts which concepts can be added to a given socket. In the example below only concepts may be added which have the value 'true' for the attribute 'visited'.
- user model variables: CRTs accessing user model variables given in this list are not allowed as 'neighbours' in the CAM.

Tab 6: DM Relations

In the sixth tab ("DM Relations", Figure 8), the relations to domain models can be specified:

- domain model relations: relations between concepts as used in domain maps can also be exploited to connect concepts in a pedagogical way. The domain model relations are given as list by adding a blank relation and modifying it in the list by clicking on it.

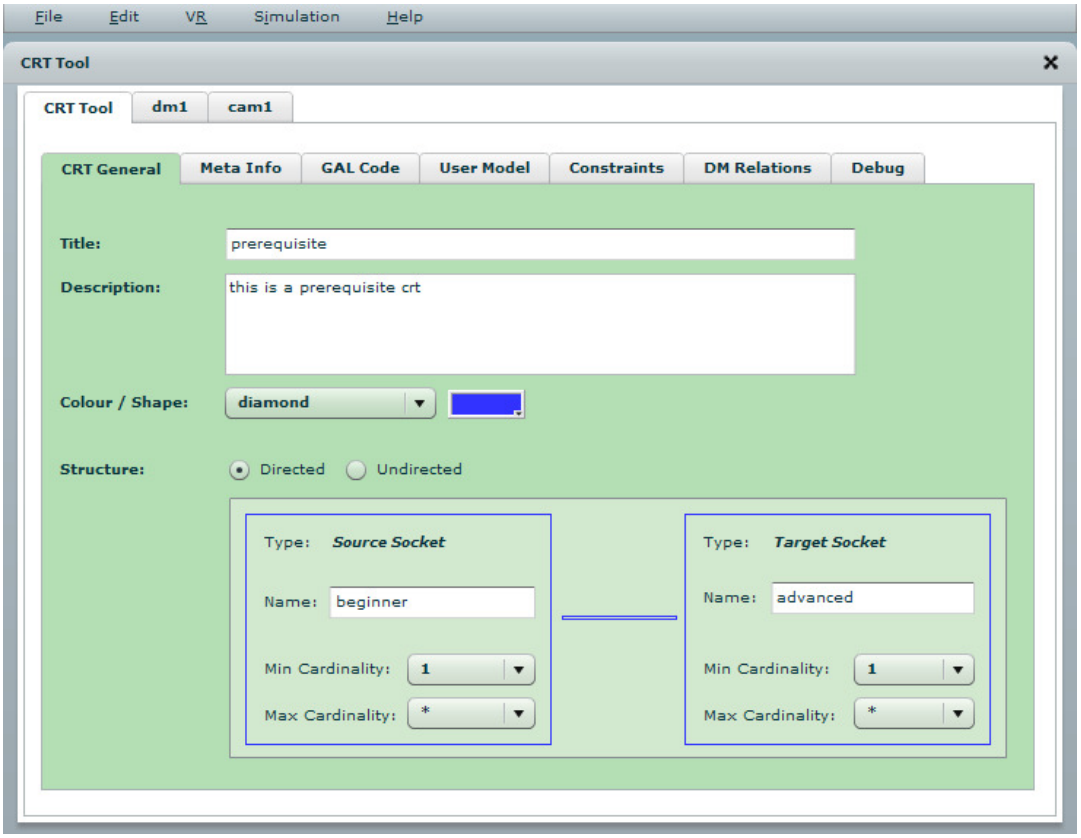

Figure 3: CRT Tool: CRT General Tab

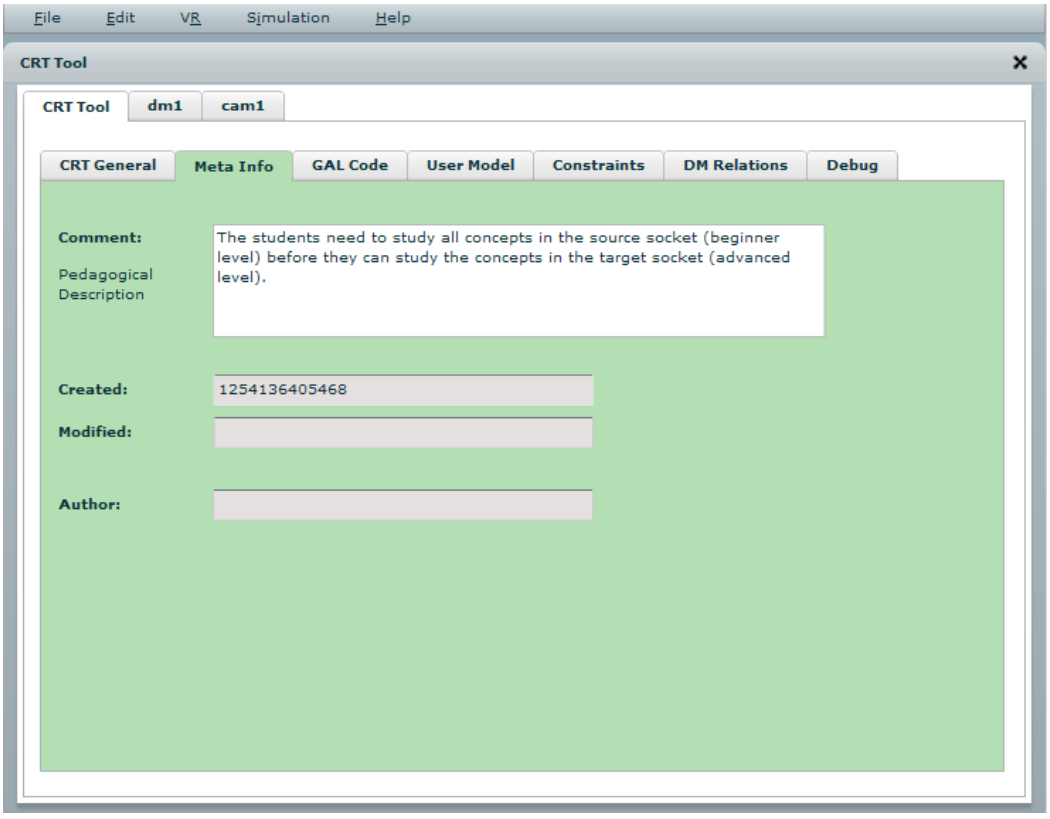

Figure 4: CRT Tool: Meta-Info Tab

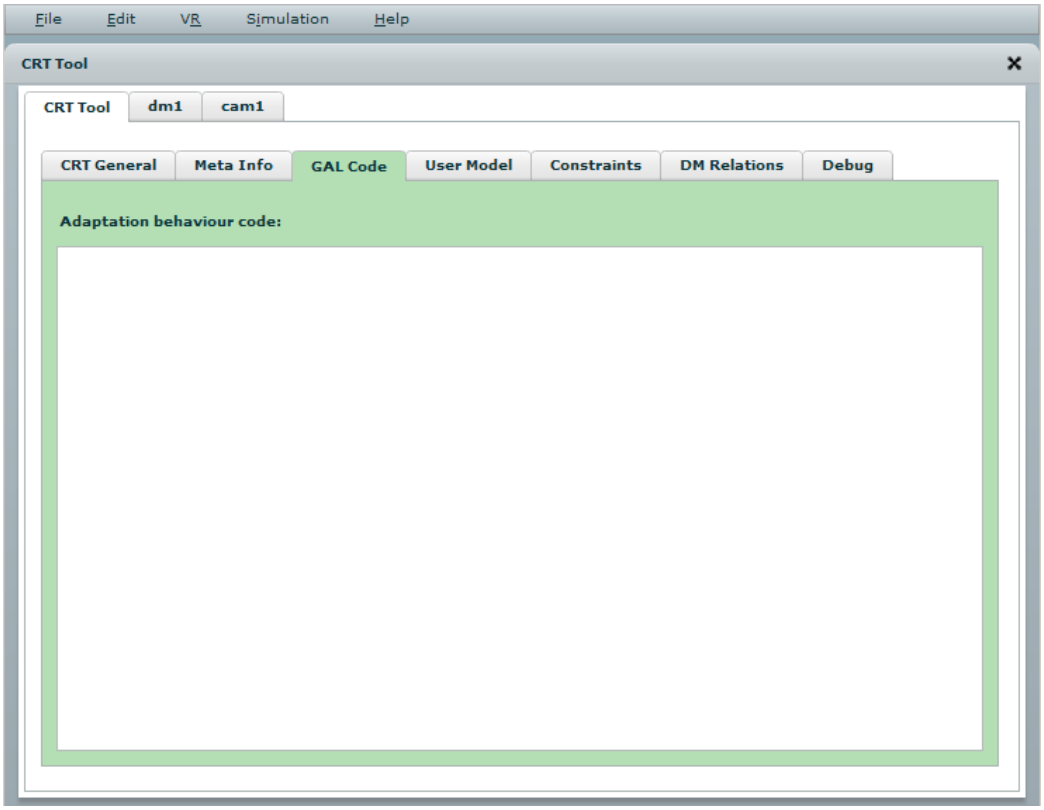

Figure 5: CRT Tool: GAL Code Tab

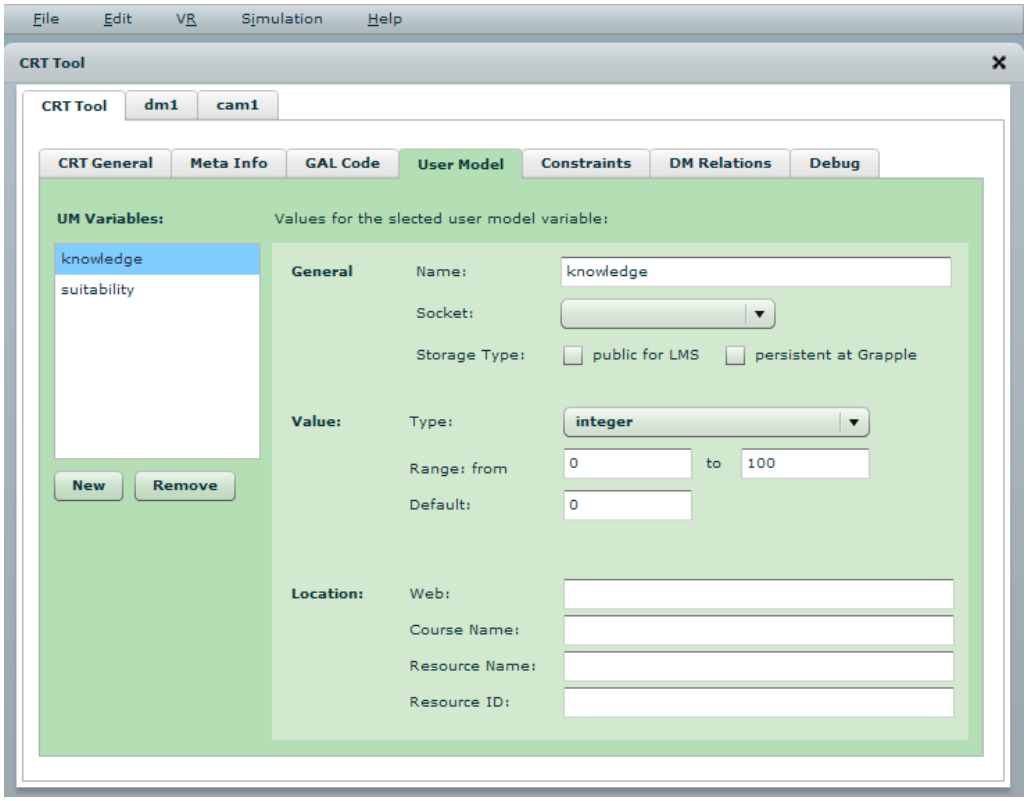

Figure 6: CRT Tool: User Model Tab

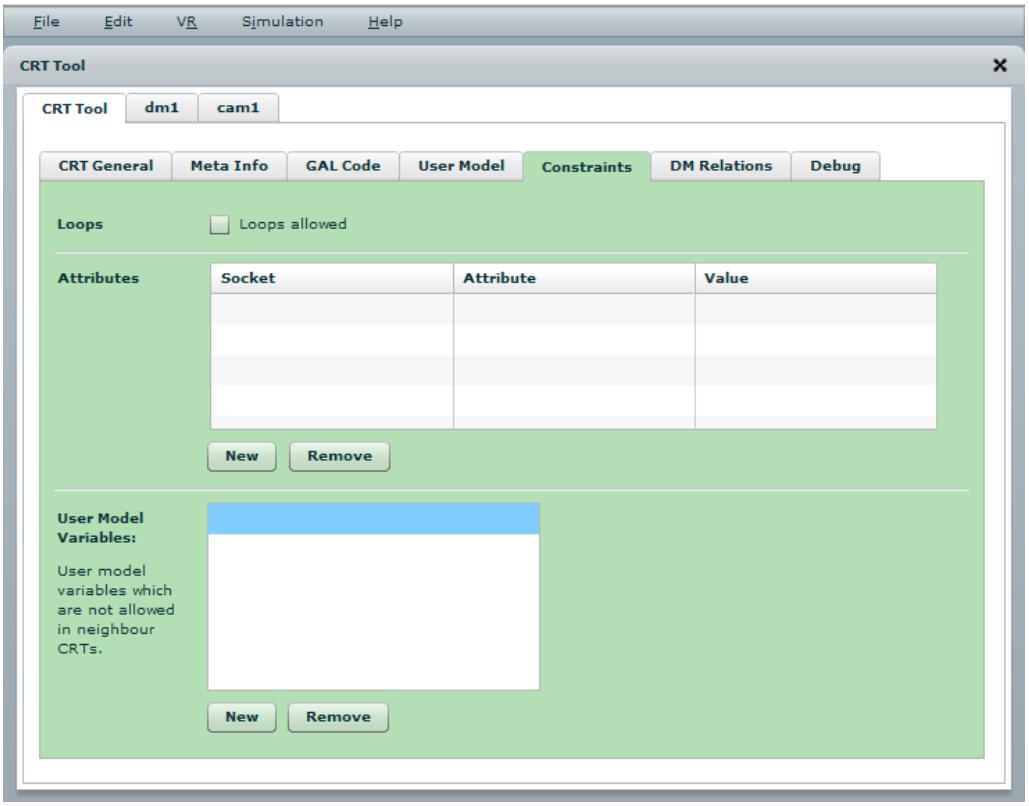

Figure 7: CRT Tool: Constraints Tab

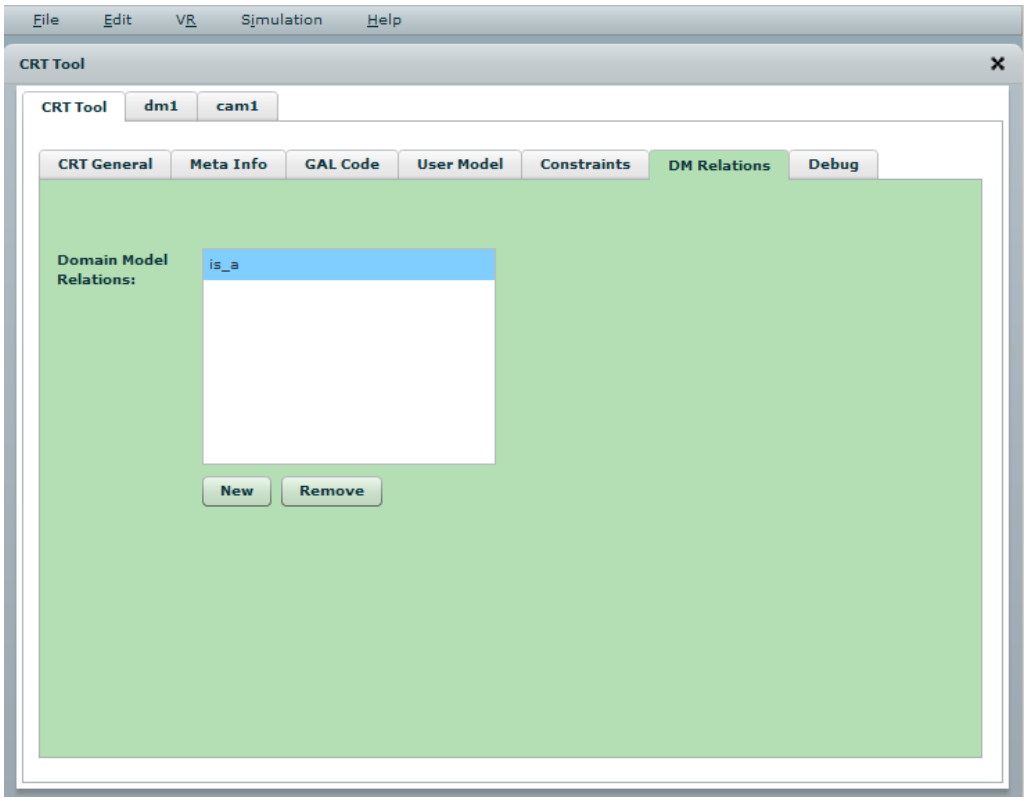

Figure 8: CRT Tool: Domain Model Relations Tab

As described in Section 3, CRTs can be loaded and saved to a Web service where they are stored 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. Opening an existing CRT is done by using the "Open" menu item (Figure 9). A list of existing CRTs will be displayed and the user can chose one and open it. Creating a new CRT can be done by using the "New" menu item (Figure 10). A dialog is displayed where the user can input title, description, and authorisation (specifying if a user has read and/or write access to this CRT). Title and description can also be changed later on. Saving a CRT is performed by using the "Save" or "Save as" menu item.

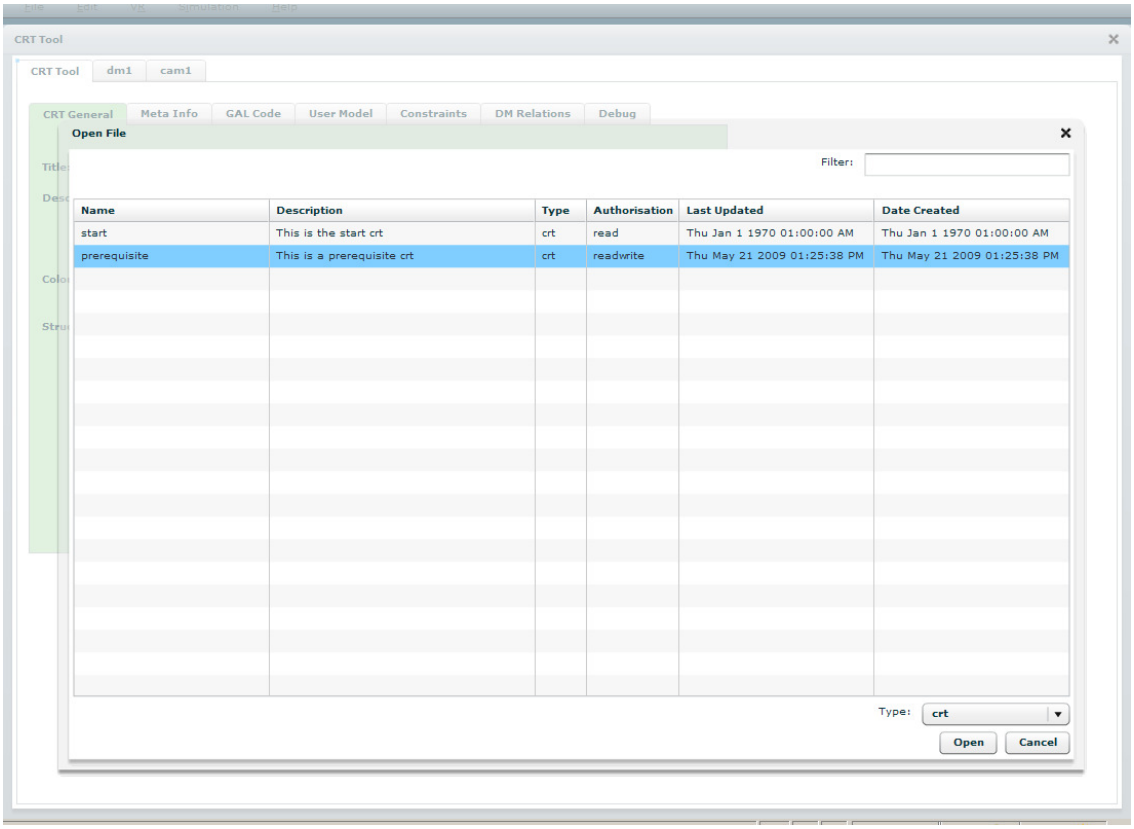

Figure 9: CRT Tool: Opening a CRT

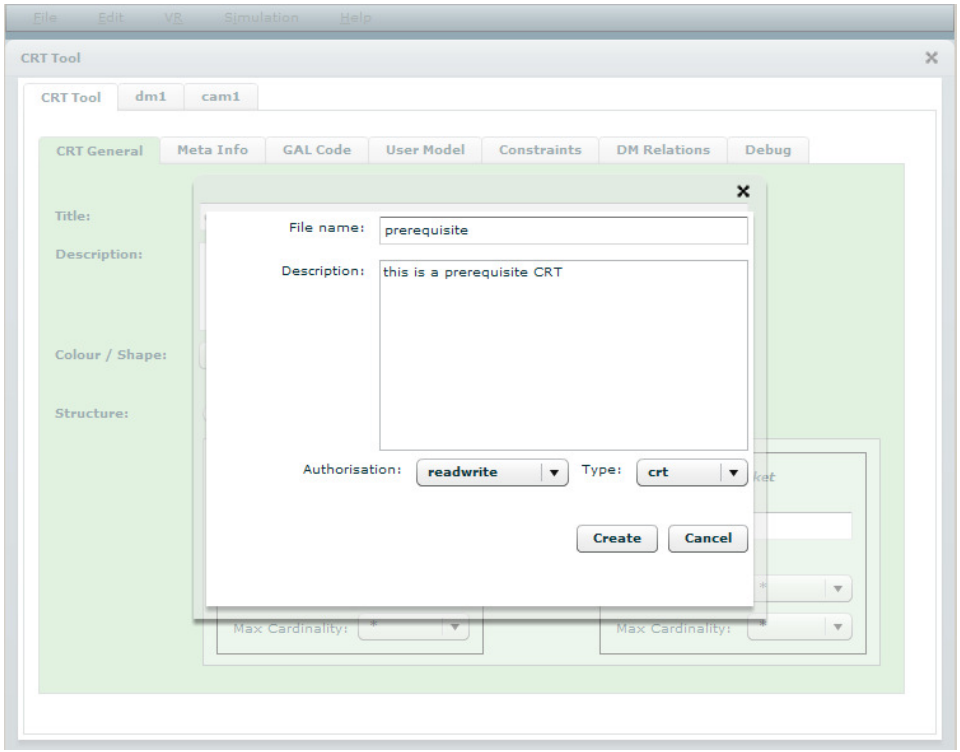

Figure 10: CRT Tool: Creating a new CRT

5 Conclusion and Outlook

This report presented a description of the initial development of the CRT Tool as it has been designed in the former Deliverable D3.2a. It has been shown 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 final version of the CRT Tool will be delivered and described in Deliverable D3.2c. Since the development of the components which make use of CRTs is not finished yet, there may be a request for further modifications of the CRT specification. The most important work to be done for next version will concern the user interface. It is aimed to improve the support for the author when creating CRTs. Furthermore, the results of the evaluation will be taken into account and influence the further work on the CRT Tool.

The CRT Tool will be demonstrated at the EC-TEL 2009 [3] conference and presented in a poster session at the ICCE 2009 [2] conference. Feedback from this demonstration and presentation sessions also might influence the further work.

References

1. Adobe Flex (2009). http://www.adobe.com/products/flex/, retrieved on 15 July 2009
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.

Appendix

5.1 CRT Examples

In this section two examples of CRT definitions are given.

The first example is a prerequisite CRT and the following XML code defines a CRT which expresses a prerequisite relation. The explanation of the example is given in Section 2.

```
<?xml version="1.0"?>

<model
  xmlns="http://grapple-project.org/GAT/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://grapple-project.org/GAT/ crt-schema.xsd">

  <header>
    <!-- the modeltype has to be 'crt' [other tools use differnt model types]-->
    <modeltype>crt</modeltype>
    <!-- the UUID of this model  -->
         <modeluuid>cf5de7f5-12b5-4720-92e5-736cac59985b</modeluuid>
    <!-- the UUID of the author of this model  -->
         <authoruuid>ffffe7f5-12b5-0000-92e5-736cac59985x</authoruuid>
    <!-- the permissions of the author of this model -->
    <authorisation>readwrite</authorisation>
    <!-- the title (name) of this CRT  -->
    <title>prerequisite</title>
    <!-- some general description of this crt  -->
    <description>This is an example of a prerequisite crt</description>
    <!-- date and time when crt was created in unix time stamp format -->
    <creationtime>1252680646515</creationtime>
    <!-- date and time when crt was moified last in unix time stamp format -->
         <updatetime>1252680646815</updatetime>
  </header>

  <body>
    <crt>

      <!--  has to be 'crt' [possible values for other tools: 'vr-crt' or 'srt'] -->
      <crtdialect>crt</crtdialect>

      <!-- free text description of the pedagogical meaning -->
      <comment>
        This CRT defines a prerequisite relations meaning that
        one concept (or socket) should be presented before another one.
      </comment>

      <!-- the visual representation: this crt is represented as red diamond -->
      <visualrepresentation>
        <shape>diamond</shape>
        <colour>0x00ff00</colour>
      </visualrepresentation>

      <!-- definition of the sockets whch define the structure of the crt -->
      <crtsockets>
        <socket type="source">
          <name>beginner</name>
          <uuid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</uuid>
          <mincardinality>1</mincardinality>
          <maxcardinality>*</maxcardinality>
          <colour>0x00FFCC</colour>
        </socket>
        <socket type="target">
          <name>advanced</name>
          <uuid>cf5de7f5-12b5-4720-92e5-pppppppppppp</uuid>
          <mincardinality>3</mincardinality>
          <maxcardinality>3</maxcardinality>
          <colour>0x00FFCC</colour>
        </socket>
      </crtsockets>

      <!-- defines the adaptive behaviour -->
      <adaptationbehaviour>
```

```
      <usermodel>
        <!-- a list of user model variables used in the gal code are listed here -->
        <umvariable>
          <!-- the name of the user model variable -->
          <umvarname>knowledge</umvarname>
          <!-- defines whether GALE needs to submit updates to GUMF or not -->
          <public>true</public>
          <!-- defines whether the value is stored or computed or retrieved -->
          <persistent>true</persistent>
          <!-- the type is float -->
          <type>float</type>
          <!-- the range of this variable is from 0.0 to 1.0 -->
          <range>
            <from>0</from>
            <to>1</to>
          </range>
          <!-- the default value of the um variable is 0 -->
          <default>0</default>
          <!-- if the concept which this um variable refers to is not located in the domain
            model, but only in an external resource, the this resource has to be specified -->
          <location>
            <!-- the link to the resource where concept and um variable is available-->
            <web>http://www.mySakaiInstallation.com/course</web>
            <remotecourse>
              <!-- the name of the resource -->
              <remotecoursename>myCourseOnSakai</remotecoursename>
              <resource>
                <resourceuniqueid>cf5de7f5-12b5-4720-92e5-ttttttttttttt</resourceuniqueid>
                <resourcename>myTest</resourcename>
              </resource>
            </remotecourse>
          </location>
        </umvariable>
      </usermodel>
      <galcode>
        <![CDATA[
          // gale code here
        ]]>
      </galcode>
    </adaptationbehaviour>



    <!-- constraints -->
    <constraints>
      <!-- expresses whether this crt can occur in a (directed) loop of
      the same crt (assumes binary crts: with 2 sockets) -->
      <allowedinloop>false</allowedinloop>
      <!-- defines a list constraints which concepts have to fulfill that they
      are allowed to be in a socket  -->
      <attributeconstraints>
        <!-- if there are multiple constraints attrConstraints they are
        connected by "and" as it is
        the more natural interpretation from an author's point of view -->
        <attrconstraint>
          <!-- in the soccet with uuid 'cf5de7f5-...' only english concepts are alloed -->
          <socketid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</socketid>
          <attributename>language</attributename>
          <requiredvalue>en</requiredvalue>
        </attrconstraint>
      </attributeconstraints>
      <!-- defines a list of UM variables that are not allowed to apear in crts
      in the "neighbourhood" (refering to at least one of the same entities) -->
      <umvariableconstraints>
        <umvariablename>visited</umvariablename>
        <umvariablename>knowledge</umvariablename>
      </umvariableconstraints>
    </constraints>

    <!-- relations to domain model: crt is referring to these domwin map relation types   -->
    <associateddmrelations>
      <relationshiptype>is_a</relationshiptype>
      <relationshiptype>part_of</relationshiptype>
    </associateddmrelations>

  </crt>
 </body>

</model>
```

The second CRT example is a definition of a goal and has a different structure than the prerequisite example.

```
<?xml version="1.0"?>

<model
  xmlns="http://grapple-project.org/GAT/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://grapple-project.org/GAT/ crt-schema.xsd">

  <header>
    <!-- the modeltype has to be 'crt' [other tools use differnt model types]-->
    <modeltype>crt</modeltype>
    <!-- the UUID of this model  -->
         <modeluuid>cf5de7f5-12b5-4720-92e5-736cac59985c</modeluuid>
    <!-- the UUID of the author of this model  -->
         <authoruuid>ffffe7f5-12b5-0000-92e5-736cac59985x</authoruuid>
    <!-- the permissions of the author of this model -->
    <authorisation>readwrite</authorisation>
    <!-- the title (name) of this CRT  -->
    <title>goal</title>
    <!-- some general description of this crt  -->
    <description>This is an example of a goal crt</description>
    <!-- date and time when crt was created in unix time stamp format -->
    <creationtime>1252680646515</creationtime>
    <!-- date and time when crt was moified last in unix time stamp format -->
         <updatetime>1252680646815</updatetime>
  </header>

  <body>
    <crt>

      <!--  has to be 'crt' [possible values for other tools: 'vr-crt' or 'srt'] -->
      <crtdialect>crt</crtdialect>

      <!-- free text description of the pedagogical meaning -->
      <comment>
        This CRT defines a goal which is a set of concepts.
      </comment>

      <!-- the visual representation: this crt is represented as red diamond -->
      <visualrepresentation>
        <shape>diamond</shape>
        <colour>0x00ff00</colour>
      </visualrepresentation>

      <!-- definition of the sockets whch define the structure of the crt -->
      <crtsockets>
        <socket type="goal">
          <name>goal</name>
          <uuid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</uuid>
          <mincardinality>1</mincardinality>
          <maxcardinality>*</maxcardinality>
          <colour>0x00FFCC</colour>
        </socket>
      </crtsockets>

      <!-- defines the adaptive behaviour -->
      <adaptationbehaviour>
        <usermodel>
        </usermodel>
        <galcode>
          <![CDATA[
            // gale code here
          ]]>
        </galcode>
      </adaptationbehaviour>



      <!-- constraints -->
      <constraints>
        <!-- expresses whether this crt can occur in a (directed) loop of
        the same crt (assumes binary crts: with 2 sockets) -->
```

```
        <allowedinloop>false</allowedinloop>
        <!-- defines a list constraints which concepts have to fulfill that they
        are allowed to be in a socket  -->
        <attributeconstraints>
          <!-- if there are multiple constraints attrConstraints they are
          connected by "and" as it is
          the more natural interpretation from an author's point of view -->
          <attrconstraint>
            <!-- in the soccet with uuid 'cf5de7f5-...' only english concepts are alloed -->
            <socketid>cf5de7f5-12b5-4720-92e5-zzzzzzzzzzzz</socketid>
            <attributename>language</attributename>
            <requiredvalue>en</requiredvalue>
          </attrconstraint>
        </attributeconstraints>
        <umvariableconstraints/>
      </constraints>

      <!-- relations to domain model: crt is referring to these domwin map relation types   -->
      <associateddmrelations/>

    </crt>
  </body>

</model>
```