OCOPOMO
Open Collaboration in Policy Modelling

D4.2 System and User Documentation

SD-2: System Documentation of CCD Tool and Annotation Extensions

Document Full Name	**OCOPOMO_D4.2-SD2_CCDTools-SysDoc.doc**
Date	**28/04/2013**
Work Package	**WP4: Integration of components**
Lead Partner	**Intersoft**
Authors	**Sabrina Scherer, Björn Lilge, Peter Bednar, Peter Smatana**
Document status	**v1.00 FINAL**
Dissemination level	**PUBLIC (PU)**

TABLE OF CONTENTS

1. INTRODUCTION AND BACKGROUND TECHNOLOGIES

This document provides the implementation and technology details of OCOPOMO plug-ins for the Eclipse platform, namely the *CCD Tool*, *Annotation Extensions* on CCD model development, *Content Repository Client*, *CCD2DRAMS* transformation tool, and *Simulation Analysis Tool*.

Installation instructions and usage guidelines for these tools are provided in the main text of D4.2 deliverable (section 3.2), as well as in the accompanying documents D4.2.-B *D4.2-B: User Manual on CCD Tools* (Scherer et al, 2013) and D4.2-D: *User Manual on Tools for Simulation Analysis and Output Scenario Generation* (Smatana and Furdik, 2013).

The main implementation technologies for developing the CCD Tool and the CCD2DRAMS Tool of the OCOPOMO toolbox are Eclipse plug-in, Eclipse Modelling Framework (EMF, http://www.eclipse.org/modeling/emf/) and Graphical Modelling Framework (GMF, http://www.eclipse.org/modeling/gmp/). For the CCD2DRAMS tool the main technology is Acceleo (http://www.eclipse.org/acceleo/). The OCOPOMO CCD Annotation Extension toolkit is based on implementation technologies such as GATE (http://gate.ac.uk/), JPedal (http://sourceforge.net/projects/jpedal/), OpenCMIS (http://chemistry.apache.org/java/opencmis.html) and REST (https://en.wikipedia.org/wiki/Representational_state_transfer). These technologies are detailed in the following subsections.

1.1. ECLIPSE PLUG-INS

The Eclipse platform has been designed to provide an Integrated Development Environment (IDE). The platform does not provide many features “by design” (Eclipse Platform Plug-in Developer Guide, 2005, p. 2). It encourages the “rapid development of integrated features based on a plug-in model” and “provides a common user interface (UI). Eclipse runs on different operation systems (OS). Plug-ins can be programmed for Eclipse independently from the underlying OS.

The Eclipse platform has an “open architecture” using “the model of a common workbench”. Plug-ins can be integrated into this workbench using “well defined extension points” (Eclipse Platform Plug-in Developer Guide, 2005). More information are available on http://www.eclipse.org.

1.2. ECLIPSE MODELLING FRAMEWORK (EMF) AND ECLIPSE GRAPHICAL MODELLING FRAMEWORK (GMF)

EMF relates modelling concepts directly to their implementations. It is a framework and code generation facility that allows it to define a model in java interfaces, UML or XML schema and generate the others. EMF is described as that is in between the "I don't need modelling" and the "Modelling rules" crowds (Steinberg, 2009).

The model used to represent models in EMF is called Ecore. The Ecore is itself an EMF model, and thus is it can be seen as its own meta-model (Steinberg, 2009). The following can be summarized:

1. Ecore and its XMI serialization is the centre of the EMF world.
2. An Ecore model can be created from any of at least three sources: a UML model, an XML Schema, or annotated Java interfaces.
3. Java implementation code and, optionally, other forms of the model can be generated from an Ecore model.

Other forms of models (in the book they take the example of an RDB schema) are possible to generate the Ecore Model (Steinberg, 2009). The CCD meta-model, presented in more details in (Scherer et al, 2013), is an Ecore model. The EMF Framework has been used in order to create the editors for a CCD.

The Eclipse Graphical Modelling Framework aims to provide a bridge between the Eclipse Modeling Framework and the Graphical Editing Framework (see at http://www.eclipse.org/gef/). EMF is a framework that provides technology to create rich graphical editors and views. GMF has been used in order to create the diagram editors.

1.3. ACCELEO

Acceleo (http://www.eclipse.org/acceleo/) is an Eclipse implementation of the "MOF Model-to-Text Language (MTL) standard, which is maintained by the Object Management Group (OMG, http://www.omg.org).

Transformation projects are created in Acceleo as Eclipse plug-ins. Based on the templates and the local specification of meta-models, an Eclipse plug-in framework is automatically generated. With an "Acceleo Launcher UI Project" the transformation can be integrated into the Eclipse user interface in a form of Eclipse plug-ins.

1.4. GATE

GATE, i.e. the General Architecture for Text Engineering (http://gate.ac.uk), is an infrastructure for developing and deploying software components that process human language (Cunningham et al, 2011). When a document is loaded into GATE, its format is analysed and converted into a GATE document, which consists of a content and one or more layers of annotation. The annotation format, internally represented as a modified form of the TIPSTER format (Grishman, 1997), is largely isomorphic with the Atlas format (Bird and Liberman, 1999) and successfully supports I/O to/from XCES and TEI (Ide, Bonhomme, and Romary, 2000). An annotation has a type, a pair of nodes pointing to positions inside the document content, and a set of attribute-values, encoding further linguistic information. Attributes are strings; values can be any Java object. An annotation layer is organised as a Directed Acyclic Graph on which the nodes are particular locations in the document content and the arcs are made out of annotations. The annotation mark-up included in the text is automatically extracted into a special annotation layer and can be used for processing or for exporting the document back to its original format.

GATE infrastructure is a rich platform for human language processing; in the OCOPOMO toolkit we are using only its part that is related to annotation management.

1.5. JPEDAL

As an Open Source library, JPedal is provided for free with source code by IDR solutions (http://www.idrsolutions.com).

Key features of JPedal are as follows:

1. Fully-featured PDF viewer with embedded font support, zooming, JBIG2 support, advanced PDF search, bookmarks, thumbnails, Layers support and more…
2. Released under the open source LGPL license with full source code for use in both commercial and Open Source projects.
3. In development for over 10 years and used in corporate software globally.
4. Upgrade route to full commercial version if additional features or support needed.
5. PDF to image converter example included ConvertPagesToImages.java.
6. Plug-ins for Eclipse, NetBeans, and IDEA.

1.6. OPENCMIS

Apache Chemistry OpenCMIS (http://chemistry.apache.org/java/opencmis.html) is a collection of Java libraries, frameworks and tools around the CMIS specification.

The goal of OpenCMIS is to make CMIS simple for Java client and server developers. It hides the binding details and provides APIs and SPIs on different abstraction levels. It also includes test tools for content repository developers and client application developers.

1.7. REST

Representational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a predominant web API design model.

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation (Fielding, 2000). Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.

The remainder of the document is organised as follows. In Chapter 2, implementation details on CCD Tool and CCD2DRAMS transformation tool are presented. The modular architecture of tools, which is extendible through further plug-ins, is shortly described. The main point of the plug-ins is the CCD Meta-model which extends the Annotation meta model. The description of both follows in section 2.4. The transformation definition of the CCD2DRAMS Tool that describes how a CCD is transformed into DRAMS code is presented in section 2.5.

CCDTool_JavaDoc.zip - see the accompanying zip package that contains the documented source code in JavaDoc for the CCD Tool.

The CCD Annotation Extensions in terms of PDF, HTML, and editable text annotations are detailed in Chapter 3. The modular plug-in architecture and interfaces of the implemented components are presented. The Content Repository Client plug-in, which enables a connection of local Eclipse environment with remote Alfresco web space, is described in section 3.4.

CCDToolsExtensions_JavaDoc.zip - see the accompanying zip package that contains the documented source code in JavaDoc for the Content Repository Client component and for the Annotation Extensions of the CCD Tools suite.

Chapter 4 presents a system documentation of the Simulation Analysis Tool, which enables a construction of output model-based scenarios on a basis of results obtained from running simulations. The Eclipse plug-in architecture is described together with interfaces of inner system components provided within the tool. Finally, a simplified sequence diagram describing the traceability mechanism, which is enabled by the Simulation Analysis Tool, is briefly explained in section 4.4.

SimulationAnalysisTool_JavaDoc.zip - see the accompanying zip package that contains the documented source code in JavaDoc for the Simulation Analysis Tool and related utilities.

2. CCD TOOL AND CCD2DRAMS TOOL

2.1. ARCHITECTUREOVERVIEW

The scheme depicted in Figure 1 presents the main Eclipse plug-ins of the CCD Tool and the CCD2DRAMS Tool.

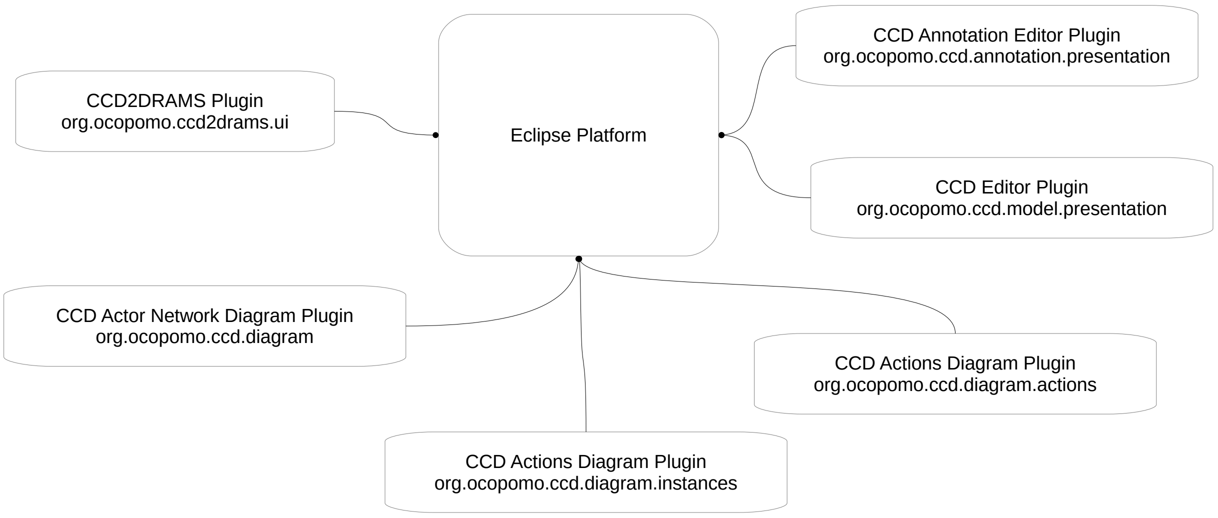
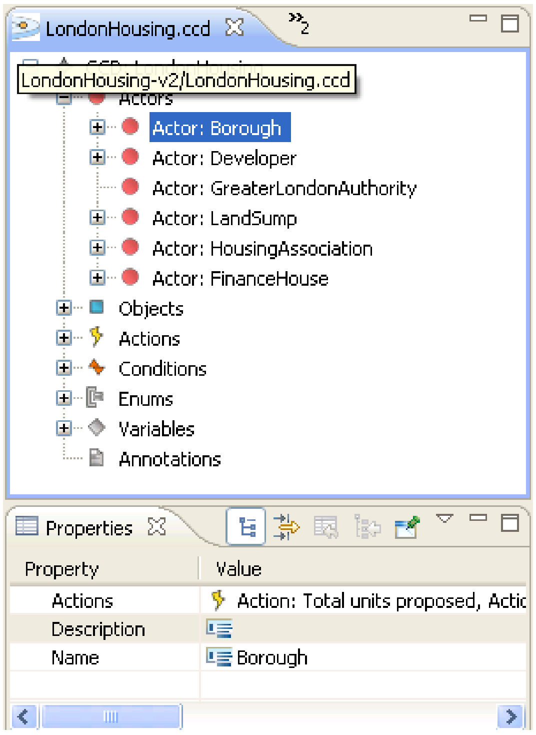

Eclipse Platform
CCD2DRAMS Plugin
org.ocopomo.ccd2drams.ui
CCD Annotation Editor Plugin
org.ocopomo.ccd.annotation.presentation
CCD Editor Plugin
org.ocopomo.ccd.model.presentation
CCD Actor Network Diagram Plugin
org.ocopomo.ccd.diagram
CCD Actions Diagram Plugin
org.ocopomo.ccd.diagram.actions
CCD Actions Diagram Plugin
org.ocopomo.ccd.diagram.instances

Figure 1: Overview of CCD and CCD 2 DRAMS Tools plug-ins.

Following, the main plug-ins are shortly described.

- CCD Editor Plug-in (`org.ocopomo.ccd.model.presentation`): Plug-in that provides user interface that allows editing of models (see Figure 2) that are using the CCD Meta-model.



Property	Value
Actions	Action: Total units proposed, Actic
Description	
Name	Borough

Figure 2: CCD Editor.

- CCD Annotation Editor Plug-in (`org.ocopomo.ccd.annoation.presentation`): Plug-in that provides user interface that allows annotation and editing of models that are using the CCD Meta-model. The Annotation Editor Plug-in (`org.ocopomo.ccd.annoation.presentation`) supports the annotation of txt files and can be extended with further plug-ins to support other file types (see section 3). It uses the CCD Editor Plug-in interfaces and classes.
- The three plug-ins to present and edit diagrams are `org.ocopomo.ccd.diagram` for actor network diagrams, `org.ocopomo.ccd.diagram.actions` for actions diagrams and `org.ocopomo.ccd.instances` for instance diagrams.
- CCD2DRAMS Plug-in (`org.ocopomo.ccd2drams.ui`): The plug-in provides the user interface to run the CCD2DRAMS transformation.

2.2. SETUP OF DEVELOPMENT ENVIRONMENT AND DEVELOPMENT CYCLE

To start developing new components or to customize CCD Tool components, the developer should install an Eclipse Indigo version. The developer should further install Plug-in Development tools, EMF SDK, GMF SDK and Acceleo SDKI from the Eclipse Project update sites. It is further necessary to include the CCD and CCD2DRAMS packages into the workspace.

2.3. CCD TOOL - CREATION OF AN EMBEDDED ANNOTATION EDITOR

The manual of the CCD Tool (Scherer et al, 2013) gives an overview of the functionalities and features implemented. This section describes how the CCD Annotation Editor can be extended for other file types than txt. Further the role of the CCD Meta-model is described and how it can be changed or adapted.

In order to create a new annotation editor (called marking component) for another type of files in the CCD annotation editor (class `AnnotationEditor`), it is necessary to implement a respective Eclipse plug-in. The plug-in should have an `Activator` class. The marking component should implement the `IMarkingEditorComponent` interface in the `org.ocopomo.annotation.presentation` package. Figure 3 shows the interfaces and classes that are contained in the `org.ocopomo.annotation.presentation` package.

In addition, the `IMarkingEditorComponent2` interface can be implemented. This interface has an additional method for better performance of annotation editor loading.

If a class implements the `IMarkingEditorComponent` interface, data needs to be further configured in the `plugin.xml` file. Therefore the `org.ocopomo.annotation.presentation.MarkingEditorComponents` extension point needs to be used. The following code fragment shows as an example the implemented marking editor component for TXT files.

The class attribute should have the name of the new implemented CCD Annotation Editor Marking Component.

```
<extension
point="org.ocopomo.annotation.presentation.MarkingEditorComponents">
      <markingEditorComponent
      class="org.ocopomo.annotation.presentation.MarkingEditor"
      fileNameExtension="txt">
      </markingEditorComponent>
</extension>
```

The component appears as a new tab in the CCD Annotation Editor if a file of the respective type is loaded.

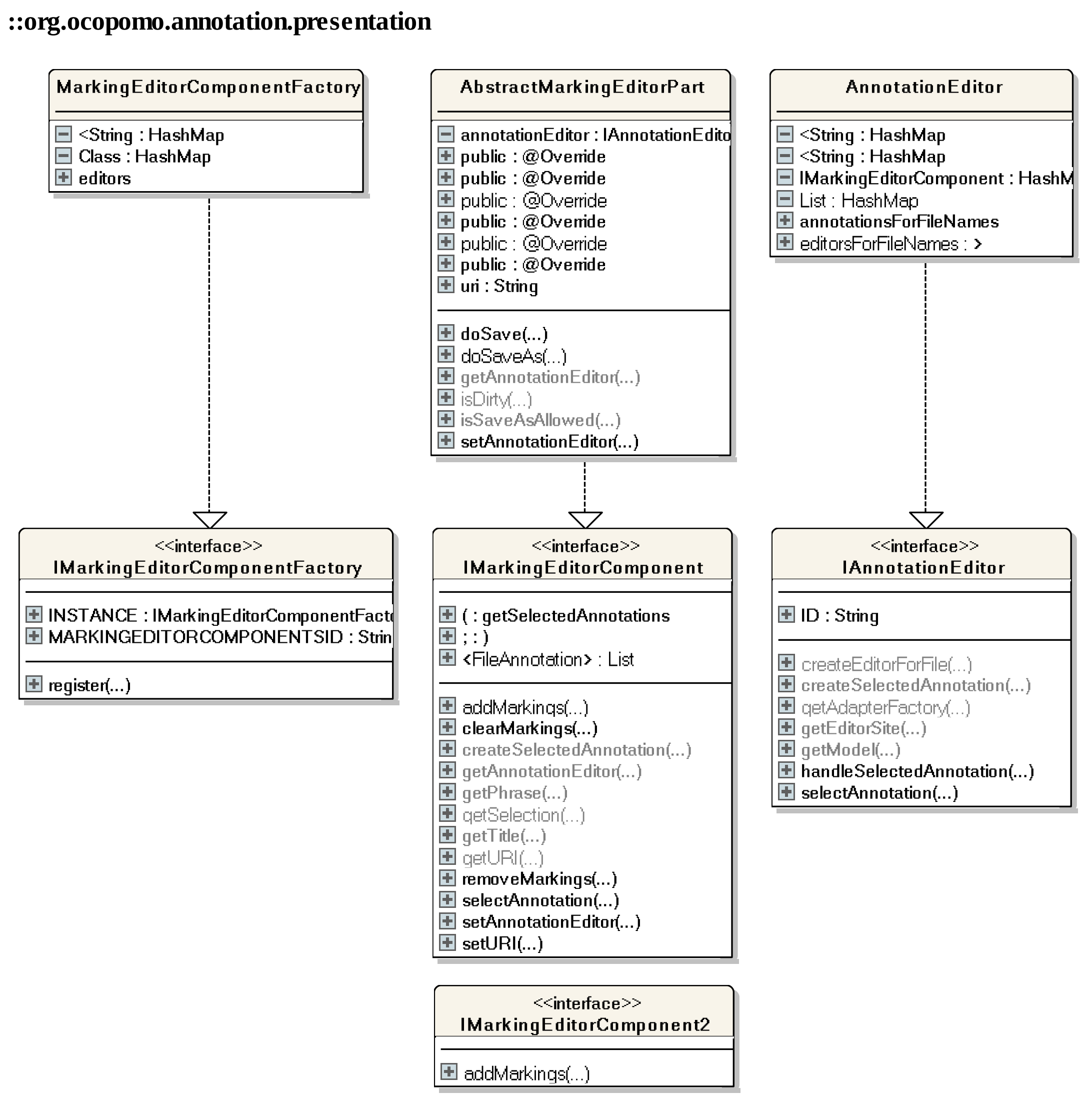


Figure 3: Interfaces and classes in `org.ocopomo.annotation.presentation` package.

2.4. CCD META-MODEL AND ANNOTATION META-MODEL

The CCD Meta-model is detailed in (Scherer et al, 2013). In order to make the Annotation components independent of the CCD Tool, the annotation meta-model is constructed as an own separate meta-model.

The structure of the annotation meta-model is depicted in Figure 4. Each CCD entity is of type “AnnotatedObject”. This way the implementation of the annotation components is independent of the particular CCD meta-model. It is just necessary to change some `xml config` files in order to allow the annotation of another Ecore meta-model.

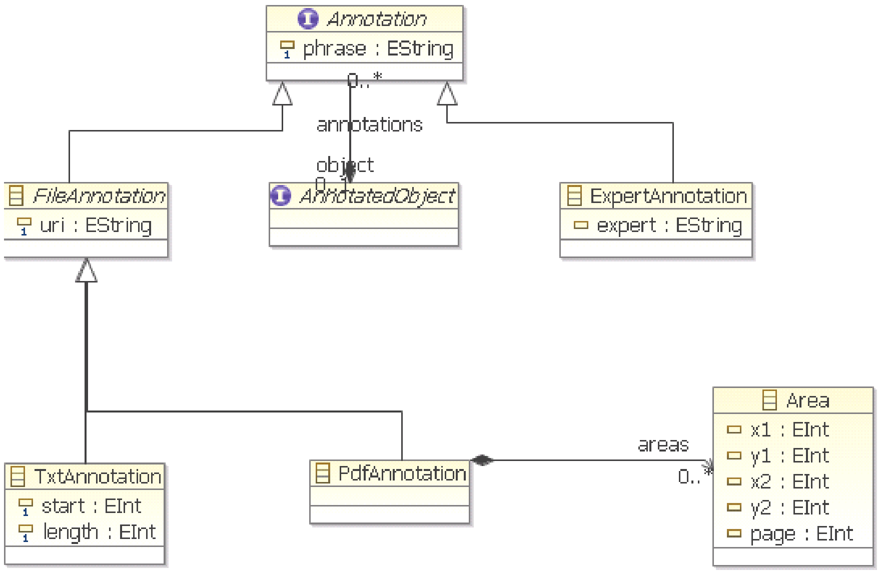
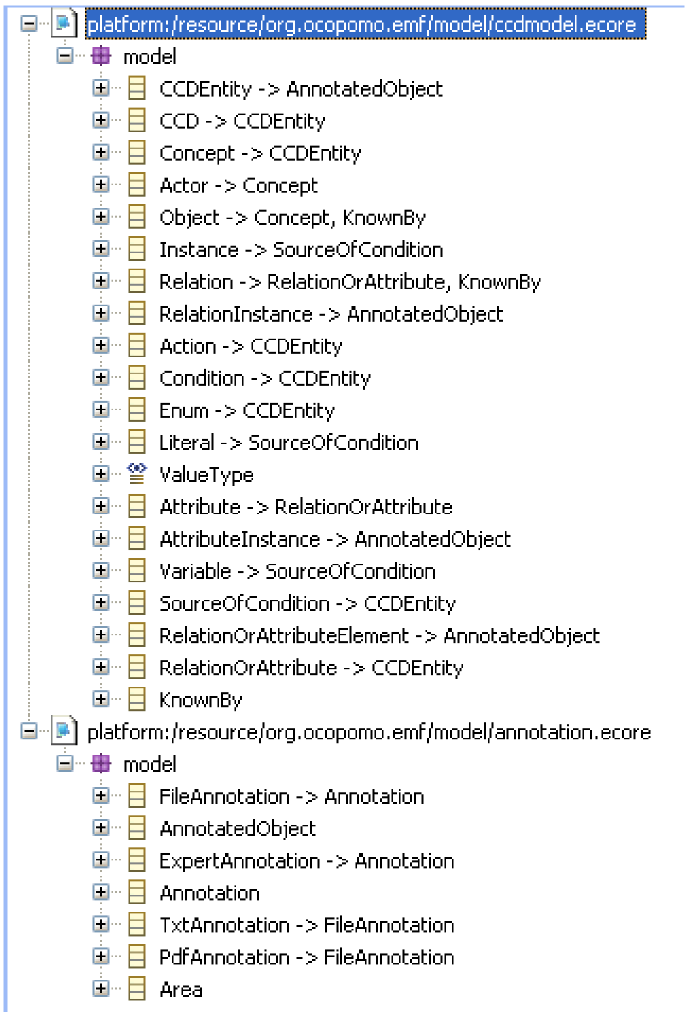


Figure 4: The annotation meta-model.

Figure 5 shows the elements of the CCD and Annotation Ecore model and their inheritance.



```
platform:/resource/org.ocopomo.emf/model/ccdmodel.ecore
  model
    CCDEntity -> AnnotatedObject
    CCD -> CCDEntity
    Concept -> CCDEntity
    Actor -> Concept
    Object -> Concept, KnownBy
    Instance -> SourceOfCondition
    Relation -> RelationOrAttribute, KnownBy
    RelationInstance -> AnnotatedObject
    Action -> CCDEntity
    Condition -> CCDEntity
    Enum -> CCDEntity
    Literal -> SourceOfCondition
    ValueType
    Attribute -> RelationOrAttribute
    AttributeInstance -> AnnotatedObject
    Variable -> SourceOfCondition
    SourceOfCondition -> CCDEntity
    RelationOrAttributeElement -> AnnotatedObject
    RelationOrAttribute -> CCDEntity
    KnownBy
platform:/resource/org.ocopomo.emf/model/annotation.ecore
  model
    FileAnnotation -> Annotation
    AnnotatedObject
    ExpertAnnotation -> Annotation
    Annotation
    TxtAnnotation -> FileAnnotation
    PdfAnnotation -> FileAnnotation
    Area
```

Figure 5: Elements of the CCD Ecore model and the Annotation Ecore model.

2.4.1. Changing the CCD Meta-Model

Changes of the CCD meta-model are to be made in the file `ccdmodel.ecore` in the project with name “org.ocopomo.emf”. The following steps need to be proceeded:

1. Change the model.
2. Reload the genmodel: ccdmodel.genmodel (available with right mouse button).
3. Open ccdmodel.genmodel and process
   a. Generate Model Code
   b. Generate Edit Code
   c. Generate Editor Code

The best way to adapt the diagrams (if necessary) is to follow the descriptions of the Eclipse GMF project and generate the diagrams.

2.4.2. Replacing the CCD Meta-Model with another Meta-Model

It is possible to use the CCD Annotation Editor with other Ecore models. Therefore the elements, of the new model, which should be able to be annotated, should implement the annotation meta-model (see Figure 4 above). How to create an Ecore model and implement respective editors is described in (Steinberg, 2009).

Then the eclipse extension point “`org.eclipse.ui.editors`” needs to be used. The name of the extension needs to be changed to the extension of the created Ecore model.

```
 <editor class="org.ocopomo.annotation.presentation.AnnotationEditor"
       contributorClass="org.ocopomo.annotation.presentation.AnnotationEdi
torActionBarContributor"
       default="true"
       extensions="ccd"
       icon="icons/ocopomo.png"
       id="org.ocopomo.annotation.presentation.AnnotationEditor"
       name="%editor.name">
</editor>
```

Each element displayed in the Annotation Editor can have an icon. Therefore for each element, the extension point “`org.eclipse.core.resources.markers`” needs to be configured:

```
<extension id="org.ocopomo.annotation.editor.marker.<nameOfTheElement>"
       name="<Full Name>"
       point="org.eclipse.core.resources.markers">
       <super type="org.ocopomo.annotation.editor.marker"></super>
</extension>
```

The following code fragment shows the configured extension point for Actor:

```
<extension id="org.ocopomo.annotation.editor.marker.actor"
       point="org.eclipse.core.resources.markers"
       name="Actor">
       <super type="org.ocopomo.annotation.editor.marker"/>
</extension>
```

In the plug-in, the icons need to be in the folder with name “icons”. Each icon needs to have the name of the respective model element (e.g. Actor.gif for the example above).

If another meta model is used, the best way to use diagrams is to follow the descriptions of the Eclipse GMF project and generate the diagrams.

2.4.3. Translate a CCD Model

In case that a CCD model has been created in different language as target stakeholders' group would like to use please use these guidelines for simple localization of CCD files:

Prerequisites:

1. Installed Java RE 6, (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
2. Installed SaxonHE9 (http://sourceforge.net/projects/saxon/files/Saxon-HE/9.4/SaxonHE9-4-0-6J.zip/download)
3. Download and unpack: http://ocopomo.ekf.tuke.sk/svn/ocopomoprj/trunk/eclipse/updatesite/TranslationXsltTemplates.zip

Translation process:

1. Convert CCD file to CSV file:
```
java  -jar saxon9he.jar -s:<input ccd file> -xsl:ccd2csv.xslt
  -o:<output csv file>
```
2. Translate CSV file to desired language (use supporting tools: MS Excel and Google translator)
3. Convert translated CSV file to XML format:
```
java -jar saxon9he.jar -it:main -xsl:csv2xml.xslt
-o:<output xml file> translated=<translated csv file>
encoding=<encoding of csv file i.e.: Windows-1250, UTF-8>
```
4. Convert original CCD file to translated CCD:
```
java -jar saxon9he.jar -s:<original input ccd file>
-xsl:ccdAndXml2ccd.xslt -o:<output translated ccd file>
translated=<xml translation from step 3>
```

Localized CCD files can be published to the remote Alfresco Data Repository following the process described in D4.2-D (Smatana and Furdik, 2013).

2.5. CCD-TO-DRAMS TRANSFORMATION DEFINITION

In order to transform a CCD, which is conforming to the CCD meta-model, into a formal simulation model, a transformation definition is necessary. Such a transformation definition has been developed for the DRAMS meta-model.

The CCD meta-model and the DRAMS meta-model have some conceptual similarities. The CCD meta-model already differentiates between the concepts *Actor* and *Object*. Actor differentiates to Object by the factor that an Actor is able to carry out an action. Thus, an Actor of the CCD-meta-model corresponds to an "Agent Class" in the DRAMS meta model.

However, the two meta-models are not equally powerful. DRAMS as a formal agent-based simulation language is more expressive than the CCD meta-model. A formal transformation definition between both meta-models was described using the "MOF 2.0 Query / View / Transformation Specification" (QVT). QVT is a standard published in 2008 by the Object Management Group (OMG, http://www.omg.org) (OMG MOFM2T, 2008). Since 2011, version 1.1 is available (OMG MOF, 2011). It consists of three languages, which form together a hybrid-transformation language. "Relations" and "Core" are two declarative languages on different abstraction levels, with a normative mapping between each other. "Operational mapping" is an imperative language, which extends both, "Relations" and "Core" languages. The "Relations" language specifies a transformation as a set of relationships between models and has a graphical concrete syntax, too. The meta-models to be transformed need to be conform to the MOF meta-meta model. Figure 6 visualizes the transformation definition between CCD meta model and DRAMS meta model using QVT.

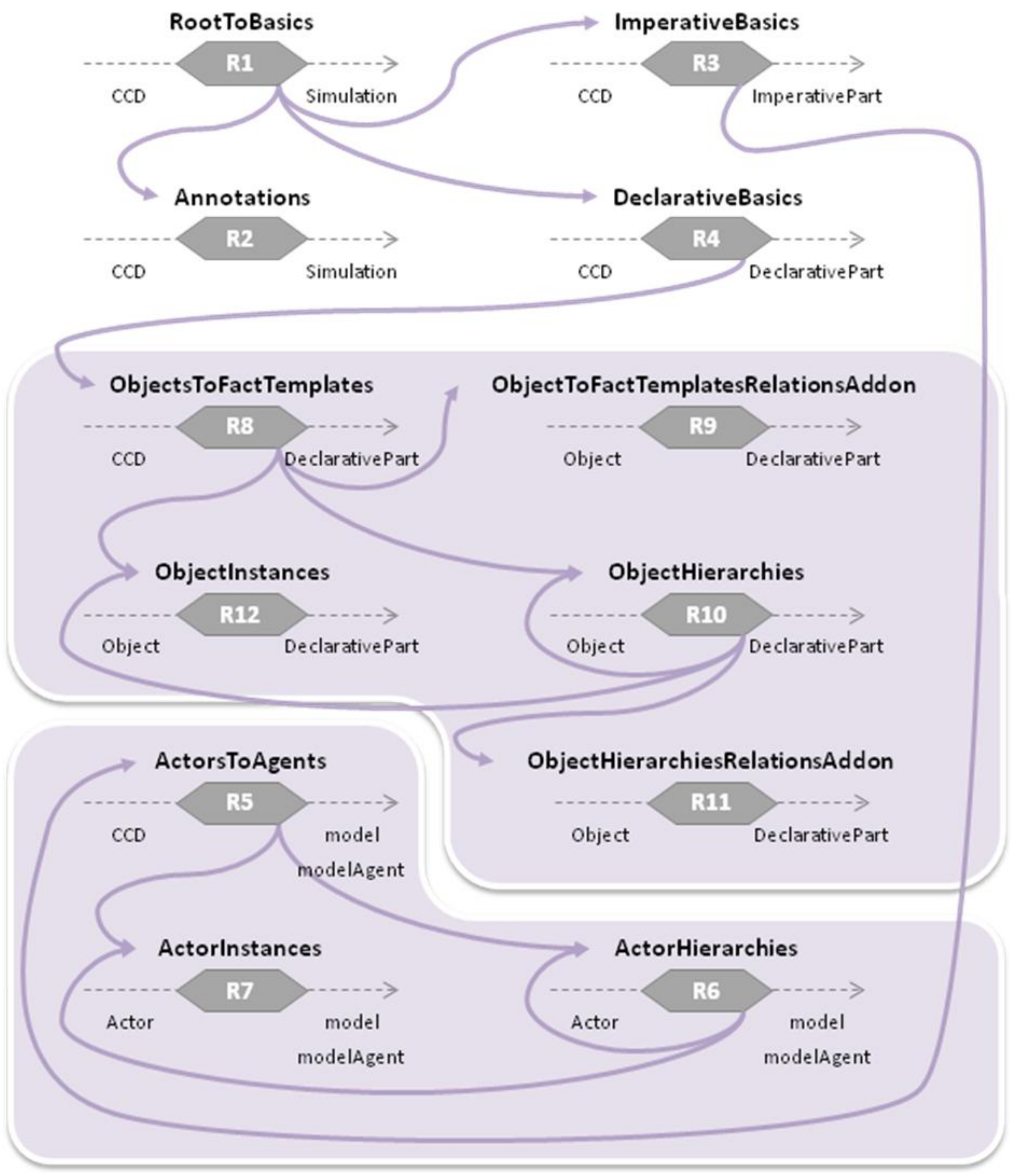
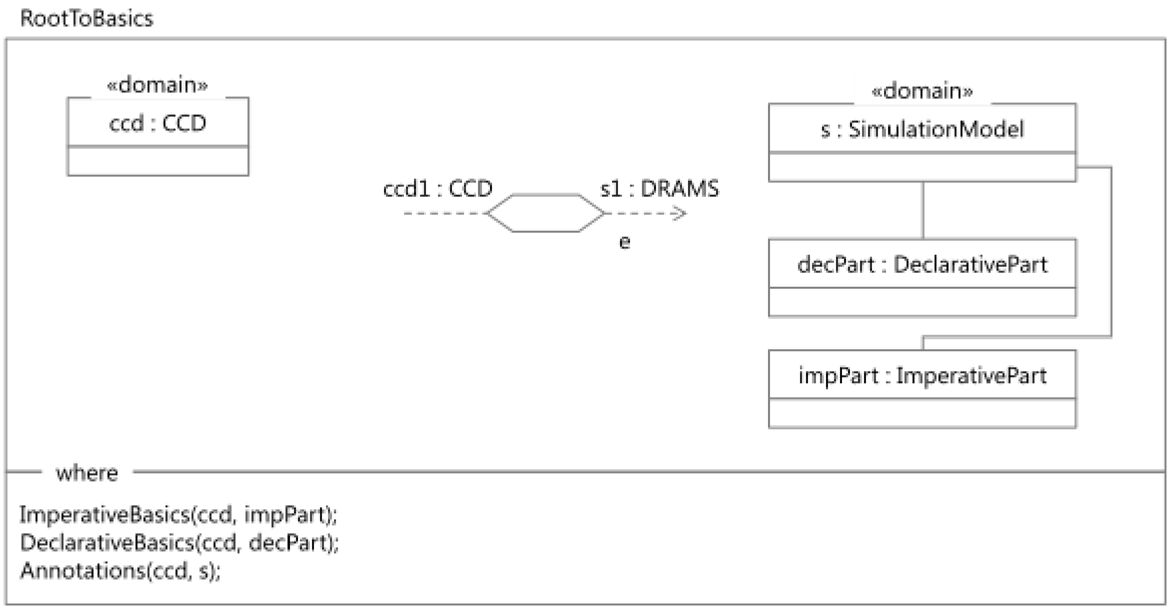


Figure 6: Transformation definition.

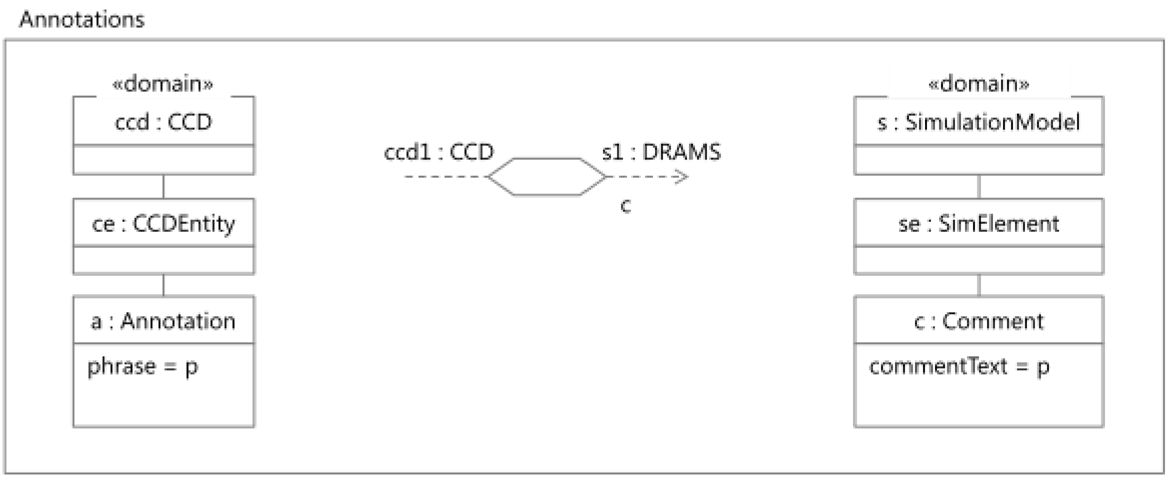
Each transformation relation has a name (e.g. *RootToBasics*), a number (e.g. *Relation 1*), a *starting domain* (left side of the relation) and an *ending domain* (right side of the relation). The “where” conditions are described with solid arrows between relations.

The starting relation *Relation 1: RootToBasics* assigns a "CCD" to a "simulation model", which consists of "DeclarativePart" and "ImparativePart" In this relation, however, no concrete elements are created, instead, this condition is referred to three other relations in the "where" condition.

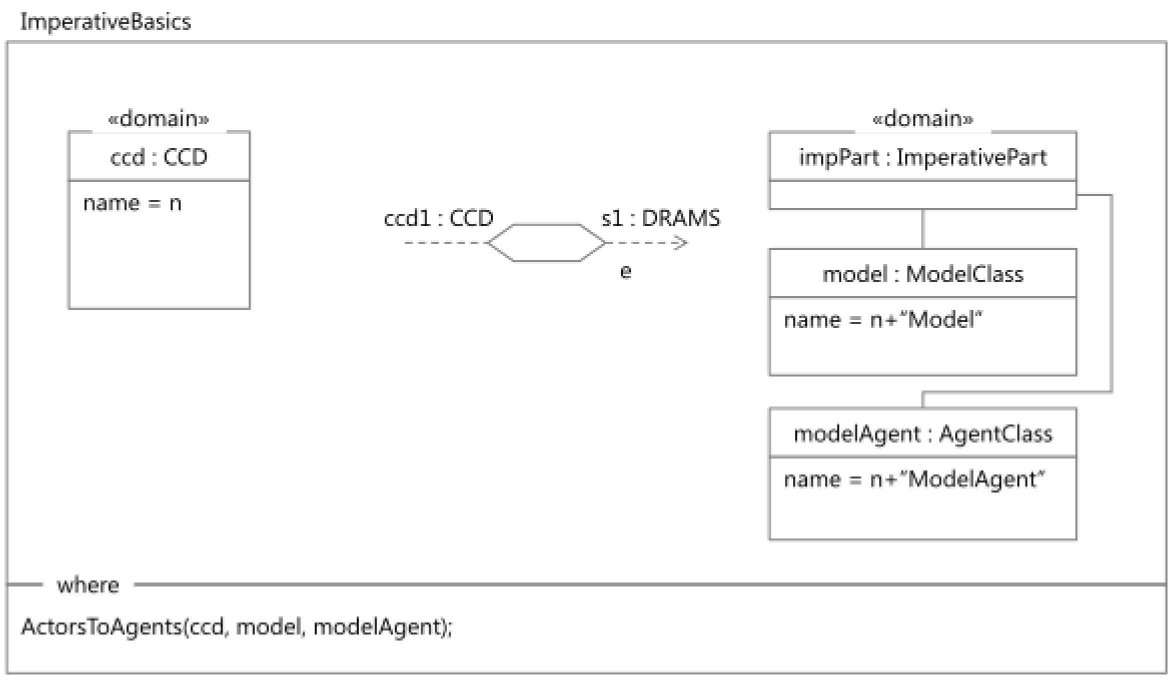


Relation 1: RootToBasics

The specification of relations in the "where" condition means that the given relations are applied to the parameters. Thus, for each "CCD" of *Relation 1: RootToBasics* the *Relation 2: Annotations* is applied. This relation transforms "annotations" of CCD elements to "Comments" in DRAMs. The value of "phrase" is used as the value of "comment text".



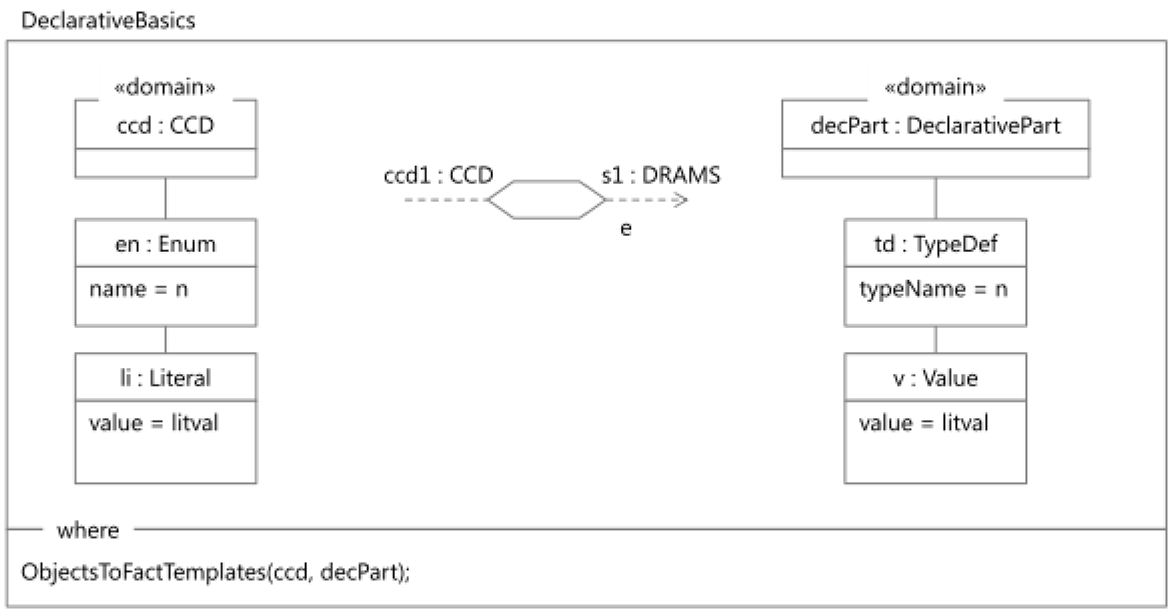
Relation 2: Annotations



Relation 3: ImperativeBasics

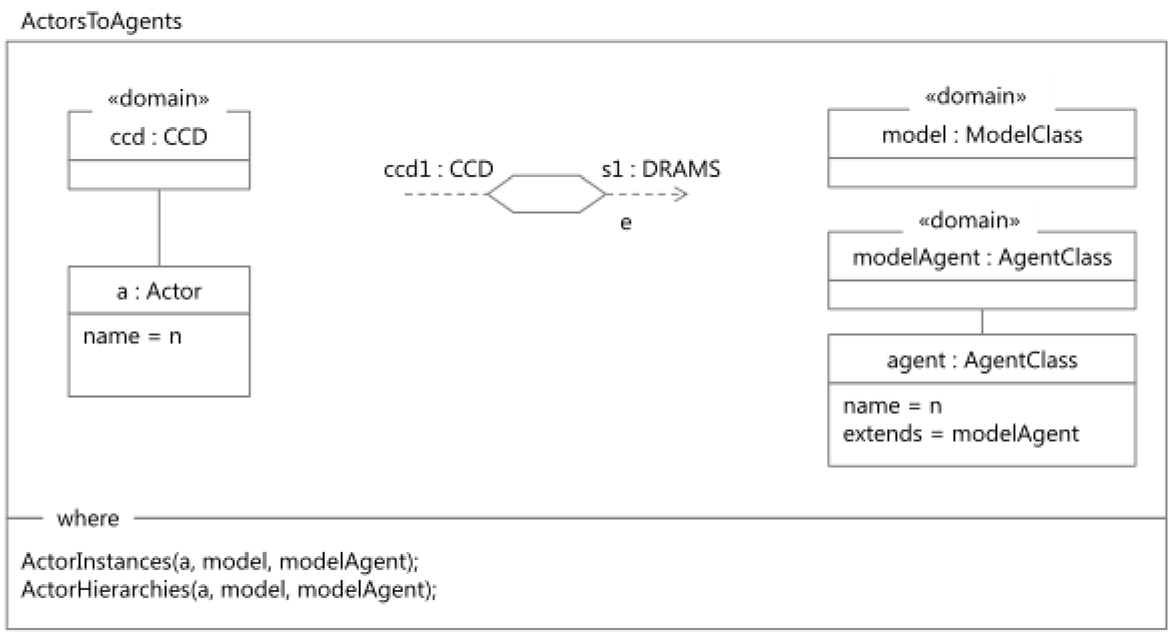
A "CCD" has a name; this name is used as part of the "model class". As *Relation 3: ImperativeBasics* shows, the abstract "agent class" also receives the name as the name prefix.

In the *Relation 4: DeclarativeBasics*, an "enum" is mapped to a "TypeDefiniton".



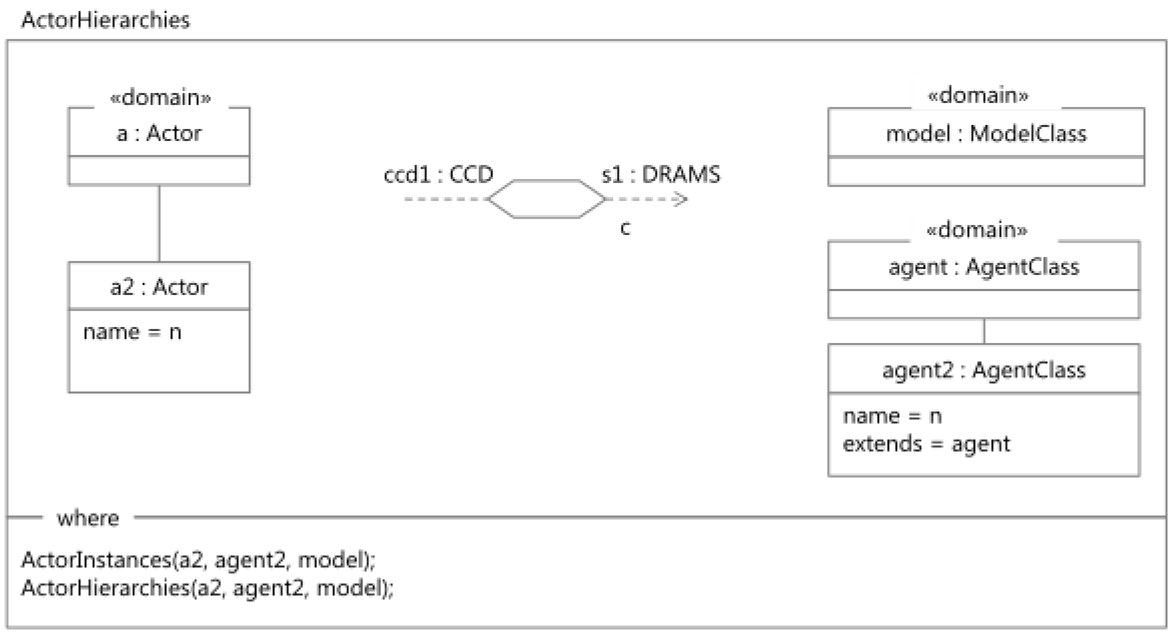
Relation 4: DeclarativeBasics

"Actors" are generated as a specialization of the abstract "agent class" "model agent".



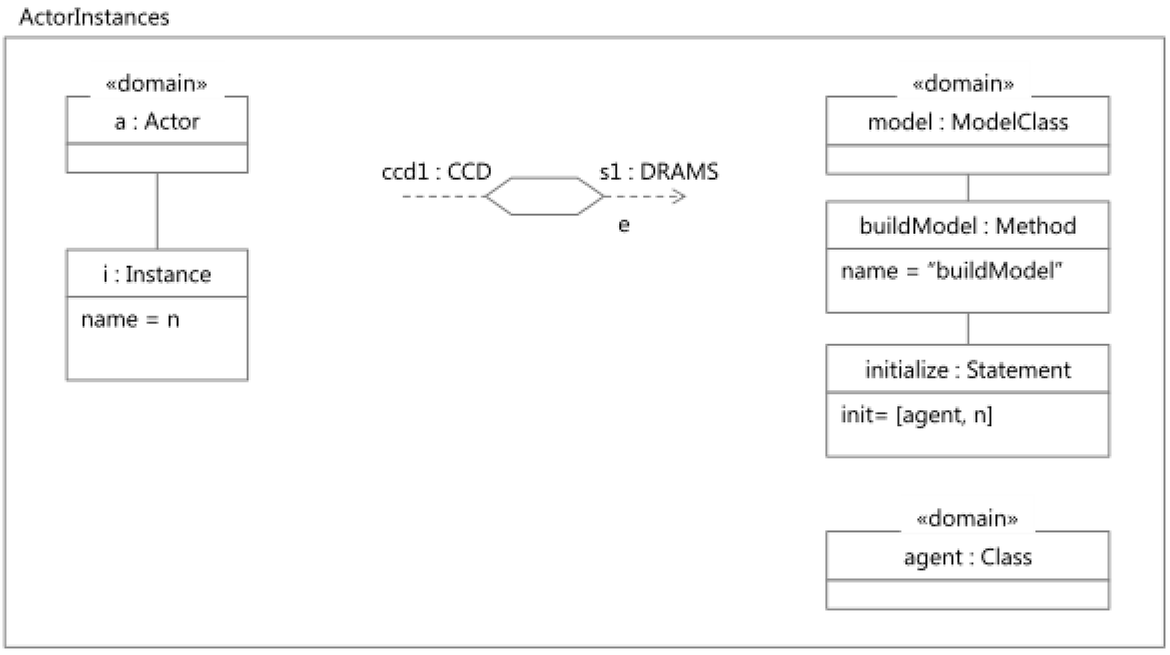
Relation 5: ActorsToAgents

With R*elation 6: ActorHierarchies*, hierarchies of "Actors" in CCD are mapped to "Agent Classes" in DRAMS.



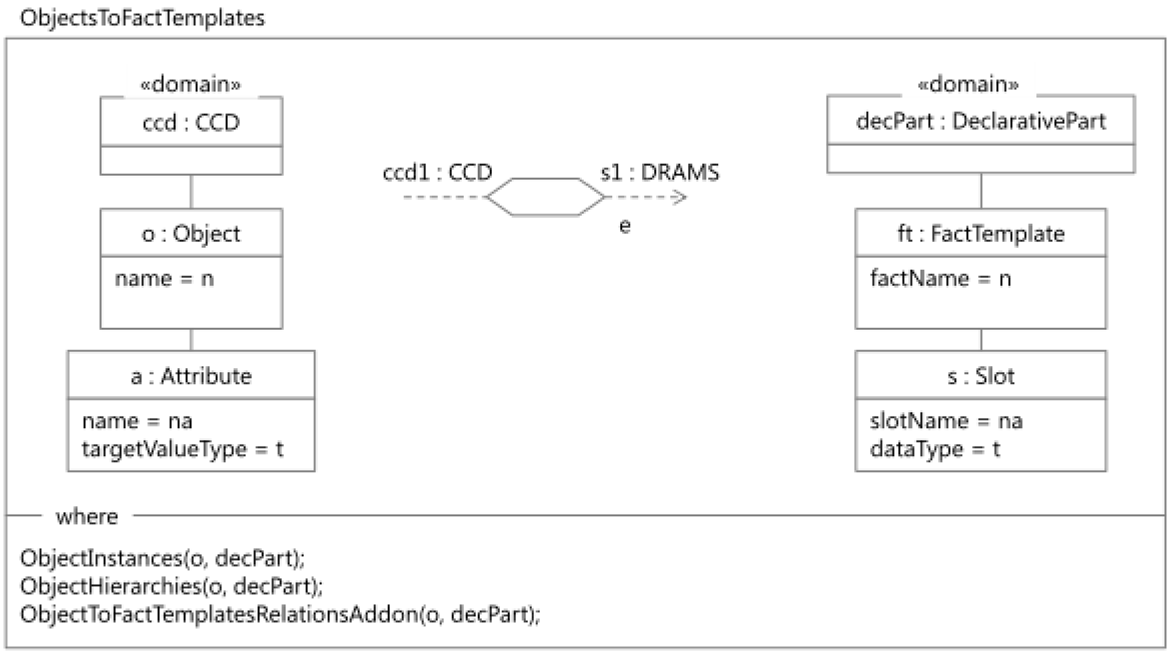
Relation 6: ActorHierarchies

For instances of "Actors", initialization statements for the respective “agent class” are created in the "build model" method in the "model class".



Relation 7: ActorInstances

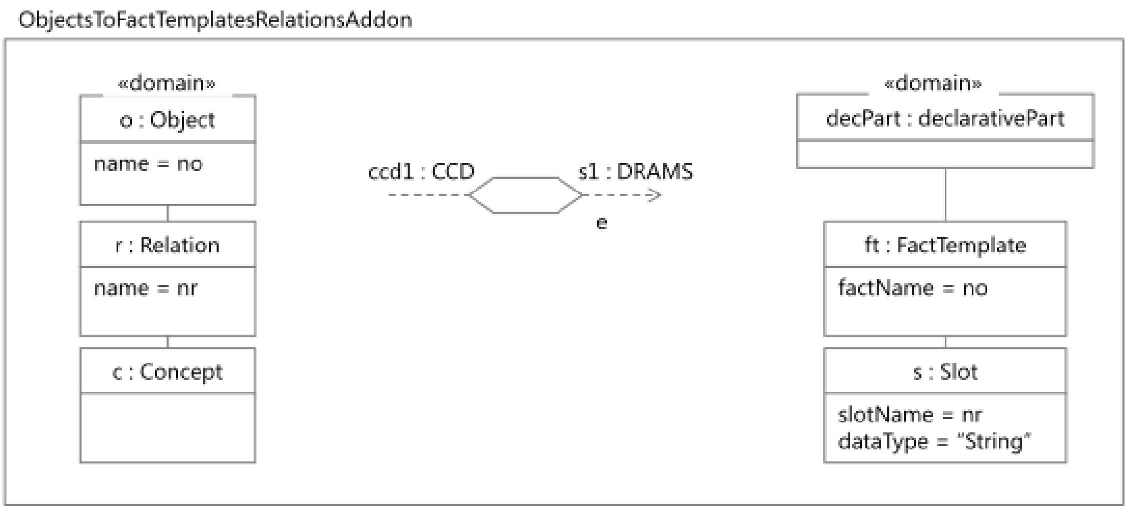
In *Relation 8: ObjectsToFactTemplates* "objects" and their "attributes" are mapped to "Fact Template" and their "slots". The fact that the choice of data types for "attribute" in CCD is a subset of the data types of "slots" in DRAMS, the data type is used directly.



Relation 8: ObjectsToFactTemplates

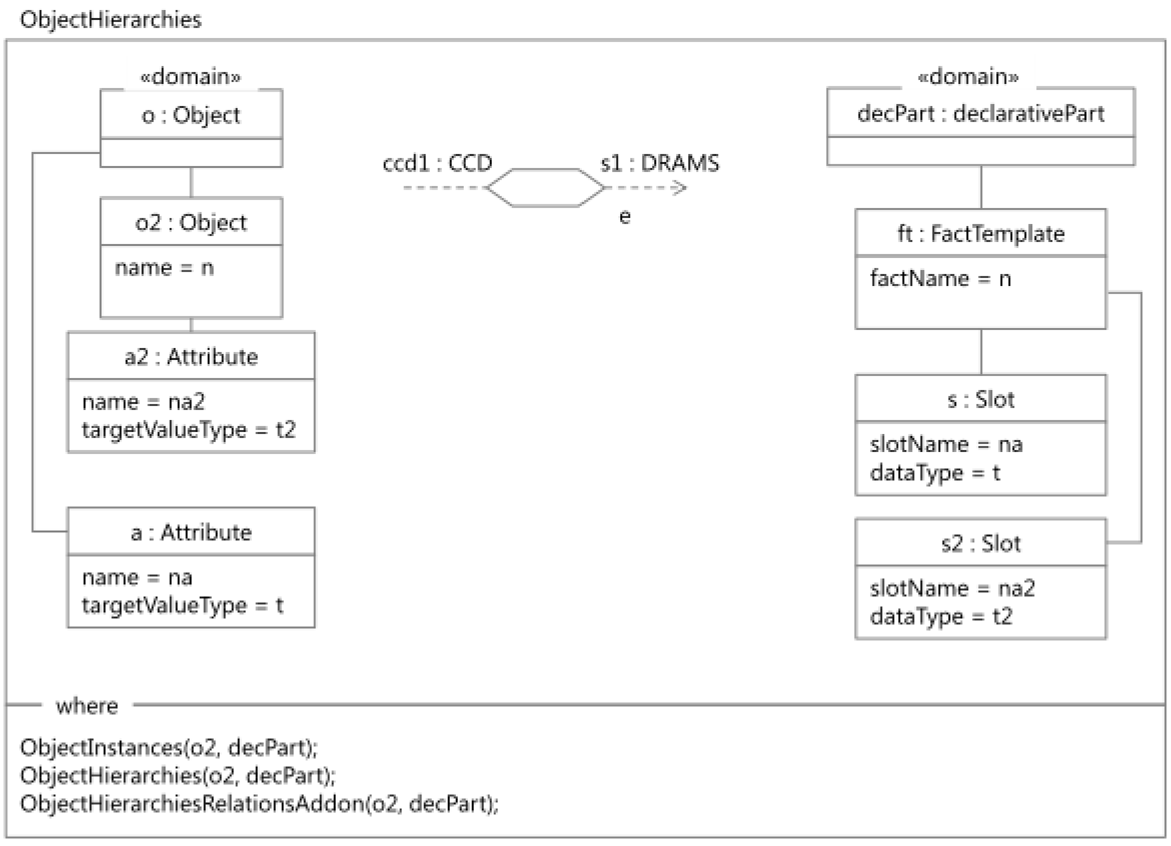
In CCD "relations" between "Concepts" are possible. In DRAMs simulation models, this is feasible without typing, therefore the correct usage at the instance level lies in the hands of the modeler.

In *Relation 9: ObjectsToFactTemplatesRelationsAddon* the name of a "Relation" of an "Object" is mapped to a "slot" in the corresponding "Fact Template". As the data type of the "slots" "string" is specified.



Relation 9: ObjectsToFactTemplatesRelationsAddon

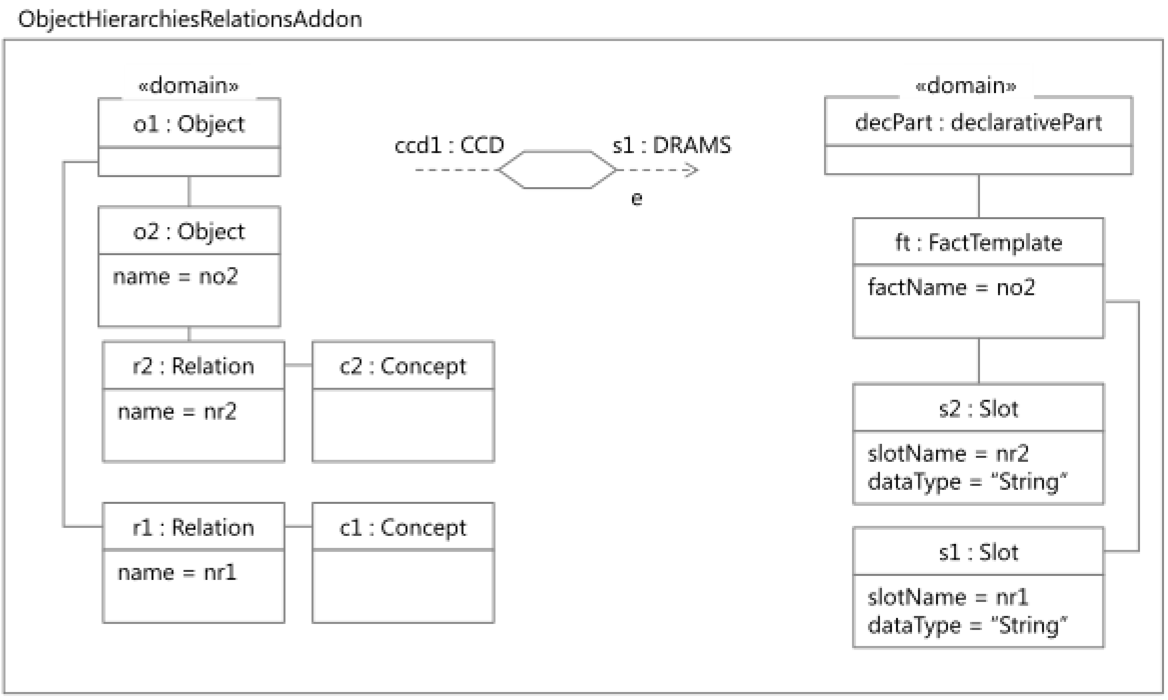
In *Relation 10: ObjectHierarchies*, hierarchies of "objects" are processed. In DRAMS, no inheritance relationships exist for "Fact Templates". Inheritance is therefore resolved by mapping. That means, all "attributes" of the generalizations of "objects" are copied into the "Fact Template". This relation is considered specifically in this form only in the direction of CCD to DRAMS.



```
ObjectInstances(o2, decPart);
ObjectHierarchies(o2, decPart);
ObjectHierarchiesRelationsAddon(o2, decPart);
```

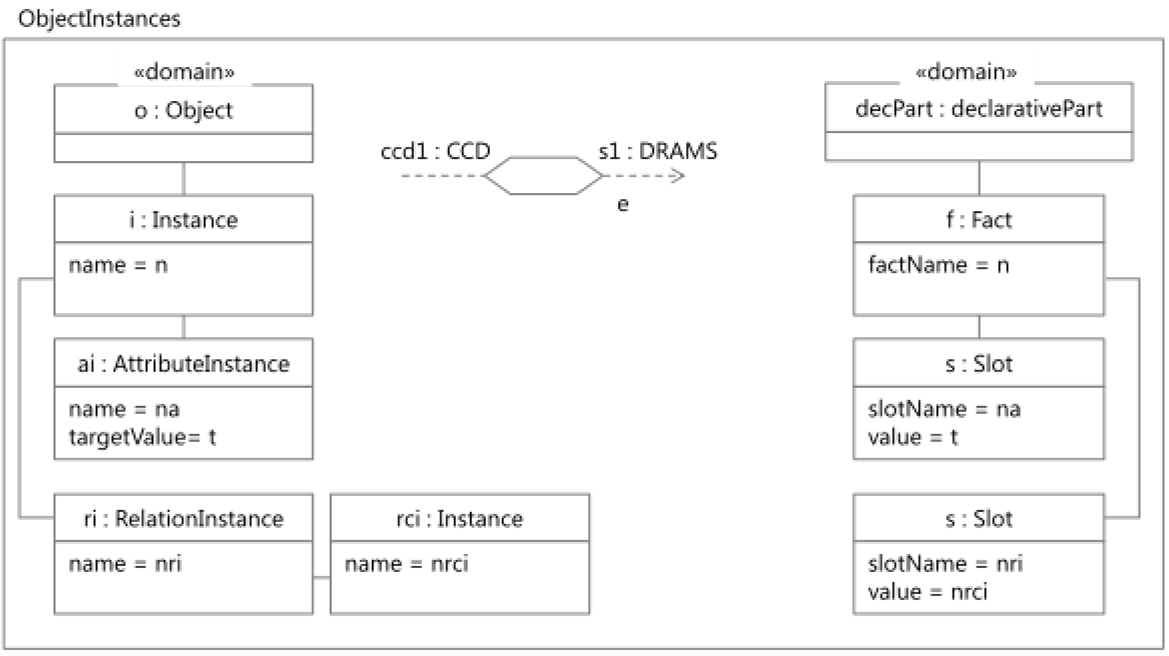
Relation 10: ObjectHierarchies

In *Relation 11: Object HierarchiesRelationsAddon* the above mentioned mapping of "Relations" for objects that lie deeper in the hierarchy is implemented.



Relation 11: ObjectHierarchiesRelationsAddon

Instances of "objects" are mapped to "facts" with the same name. Both name and data type of "attributes" are accordingly for "slots". The relationship described above from "Objects" with "Concepts" on "Relations" works here at the level of instances. Accordingly, the "RelationInstance" is the name for "slot name" and of the "Instance" is the name for "value" of "slots".



Relation 12: ObjectInstances

3. CCD ANNOTATION EXTENSIONS AND CONTENT REPOSITORY CLIENT

3.1. ARCHITECTURE OVERVIEW

The scheme depicted in Figure 7 presents the main plug-ins of the CCD Annotation Extensions and Content Repository Client.

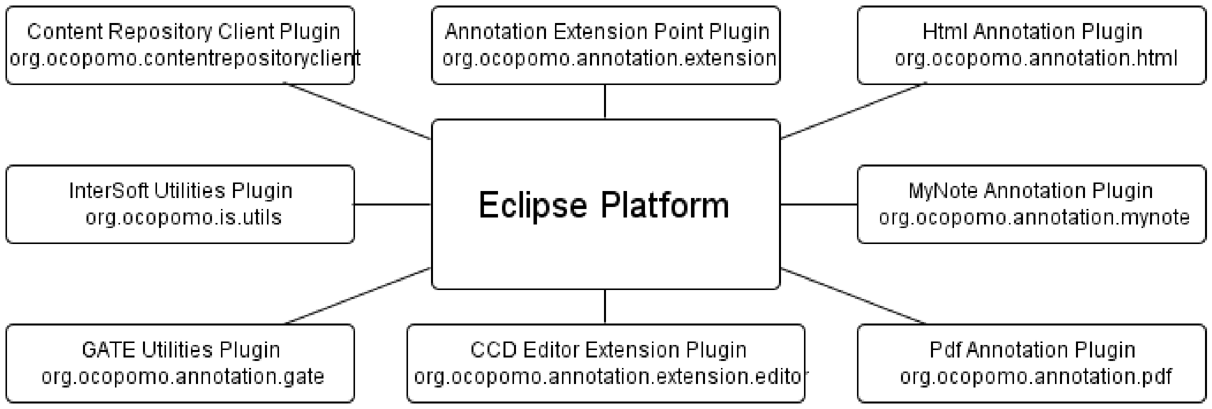

Content Repository Client Plugin
org.ocopomo.contentrepositoryclient
Annotation Extension Point Plugin
org.ocopomo.annotation.extension
Html Annotation Plugin
org.ocopomo.annotation.html
InterSoft Utilities Plugin
org.ocopomo.is.utils
Eclipse Platform
MyNote Annotation Plugin
org.ocopomo.annotation.mynote
GATE Utilities Plugin
org.ocopomo.annotation.gate
CCD Editor Extension Plugin
org.ocopomo.annotation.extension.editor
Pdf Annotation Plugin
org.ocopomo.annotation.pdf

Figure 7: Overview of CCD Annotation Extensions and Content Repository Client plug-ins.

Following, the main plug-ins are shortly described.

- Annotation Extension Point Plug-in (`org.ocopomo.annotation.extension`): Plug-in that provides extension point for creation of new annotation extension (new file types or new functionality).
- Html Annotation Plug-in (`org.ocopomo.annotation.html`): Plug-in that is used for annotation of HTML files (mainly Alfresco wiki pages) onto CCD model.
- MyNote Annotation Plug-in (`org.ocopomo.annotation.mynote`): Plug-in that is used for annotation of plain text onto CCD model. Plain text file could be modified in process of annotation.
- Pdf Annotation Plug-in (`org.ocopomo.annotation.pdf`): Plug-in that is used for annotation of PDF files onto CCD model.
- CCD Editor Extension Plug-in (`org.ocopomo.annotation.extension.editor`): Plug-in that extends of CCD Editor Plug-in. This plug-in is able to recognize all annotators that has implemented Annotation Extension Point.
- InterSoft Utilities Plug-in (`org.ocopomo.is.utils`): Plug-in that provides utilities for parsing specific XML files and logging support.
- GATE Utilities Plug-in (`org.ocopomo.annotation.gate`): Plug-in that wraps GATE core library and provides API for initialization and usage of GATE annotation management.
- Content Repository Client Plugin (`org.ocopomo.contentrepositoryclient`): Plug-in that provides API and GUI for remote accessing of Alfresco content repository via CMIS and REST.

3.2. SETUP OF DEVELOPMENT ENVIRONMENT AND DEVELOPMENT CYCLE

To start developing new components or to customize CCD Annotation Extension components, the developer should install an Eclipse Indigo version. The developer should further install PLUGIN Development tools and CCD Tools plug-ins from the OCOPOMO and Eclipse Project update sites. It is further necessary to include the CCD Annotation Extensions and Content Repository Client packages into the workspace.

3.3. CCD ANNOTATION EXTENSIONS - INTERFACE DESCRIPTION

It is designed as an extension point *org.ocopomo.annotator* that could be implemented by all annotation extensions. Annotation extension is then resolved by CCD Editor Extension and editor is able to accept these new file types associated with new annotation extension and display them. New Eclipse extension point extends *CCDTextAnnotator* (see Figure 8).

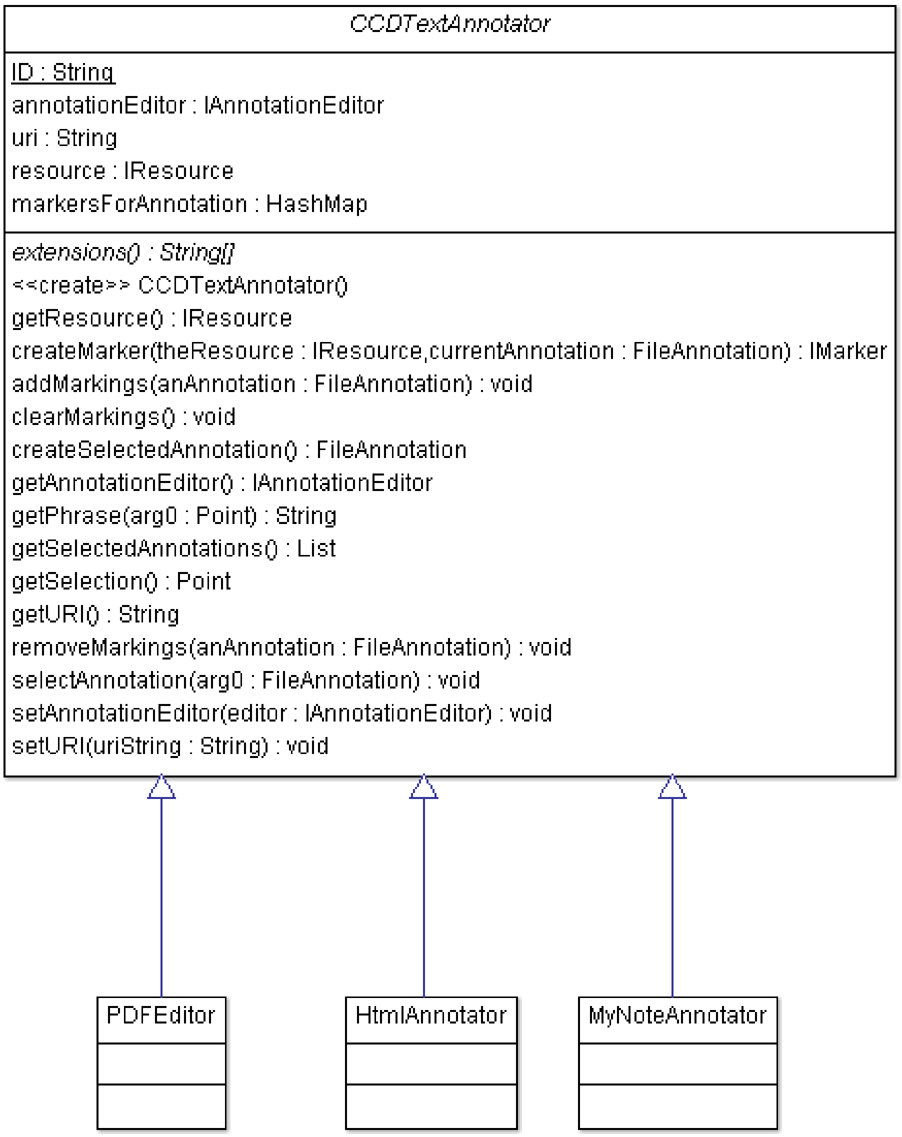

CCDTextAnnotator
ID : String
annotationEditor : IAnnotationEditor
uri : String
resource : IResource
markersForAnnotation : HashMap
extensions() : String[]
<<create>> CCDTextAnnotator()
getResource() : IResource
createMarker(theResource : IResource,currentAnnotation : FileAnnotation) : IMarker
addMarkings(anAnnotation : FileAnnotation) : void
clearMarkings() : void
createSelectedAnnotation() : FileAnnotation
getAnnotationEditor() : IAnnotationEditor
getPhrase(arg0 : Point) : String
getSelectedAnnotations() : List
getSelection() : Point
getURI() : String
removeMarkings(anAnnotation : FileAnnotation) : void
selectAnnotation(arg0 : FileAnnotation) : void
setAnnotationEditor(editor : IAnnotationEditor) : void
setURI(uriString : String) : void
PDFEditor
HtmlAnnotator
MyNoteAnnotator

Figure 8: Hierarchy of Annotation Extensions.

3.3.1. PDF Annotator

PDF Annotator incorporated JPedal which is open source PDF viewer. Updated `org.jpedal` library is used for displaying and annotation of PDF files. Original library is not able to handle annotation therefore we were implemented some modifications of the library to support displaying of annotation in PDF files. The major modifications are:

- Class `org.jpedal.PdfAnnotation` used for storing annotations (see Figure 9)
- `org.jpedal.SingleDisplay` method `drawPage(AffineTransform viewScaling, AffineTransform displayScaling, int pageUsedForTransform)`*: Rectangle* should display listed annotations
- *`org.jpedal.PdfDecoder`* main class that is used for creation of PDF Viewer

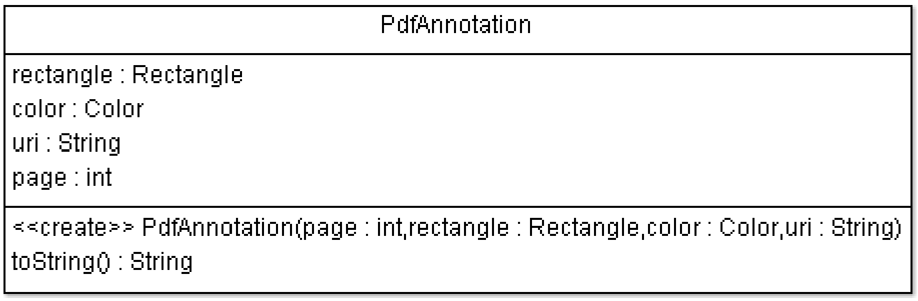


Figure 9: PdfAnnotation class.

All PDF annotations are stored as reference to individual page and areas defined as rectangles from which that annotations are consisted. Individual rectangle areas are defined as absolute position of top-left pixel and as absolute position bottom-right pixel.

3.3.2. HTML Annotator

Core of HTML annotator is `javax.swing.JTextPane` component. This component models paragraphs that are composed of runs of character level attributes. Each paragraph may have a logical style attached to it which contains the default attributes to use if not overridden by attributes set on the paragraph or character run. Components and images may be embedded in the flow of text. It also supports HTML file formats.

3.3.3. MyNote Annotator

MyNote Annotator is a plug-in that allows to annotate plain text files onto CCD model. These plain text files could be modified during annotation process. files annotated by the MyNote Annotator are displayed as plain text but they are XML files containing tags that allows referencing text parts via CCD model. Referencing by start character and length is unsuccessful because text could be modified and CCD model will reference to wrong area.

XML files are created and managed by GATE library. Displaying of plain text is managed by `org.eclipse.jface.text.source.projection.ProjectionViewer` which allows to hide Master XML Document from users and display only its projection (see Figure 10).

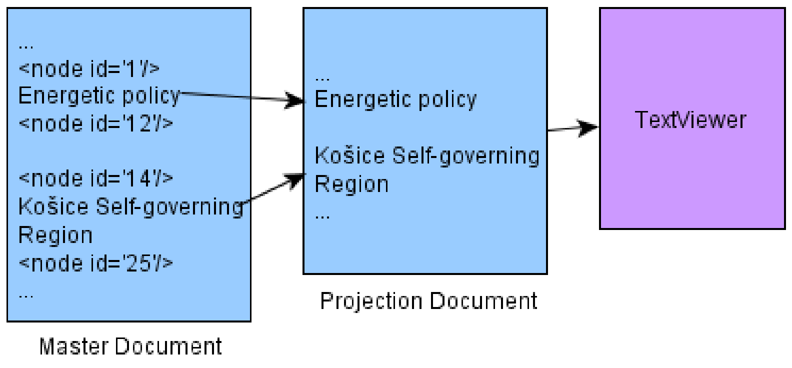


Figure 10: Basic principle of projection Viewer.

3.3.4. CCD Editor Extension

CCD Editor Extension extends the CCD Editor plug-in and it is able to recognize all annotation extensions that extend *org.ocopomo.annotation* extension point.

3.4. CONTENT REPOSITORY CLIENT

The Content Repository Client component is developed as a graphical extension of CMIS and REST protocols and as an internal high-level API for communication between a remote Alfresco content repository and local Eclipse components. For example, Simulation Analysis Tool, described in Chapter 4 below, uses it for publishing output model-based scenarios (see also in D4.2-D (Smatana and Furdik, 2013)). CCD Tool uses the Content Repository Client mainly for annotation of a content located in the Alfresco content repository.

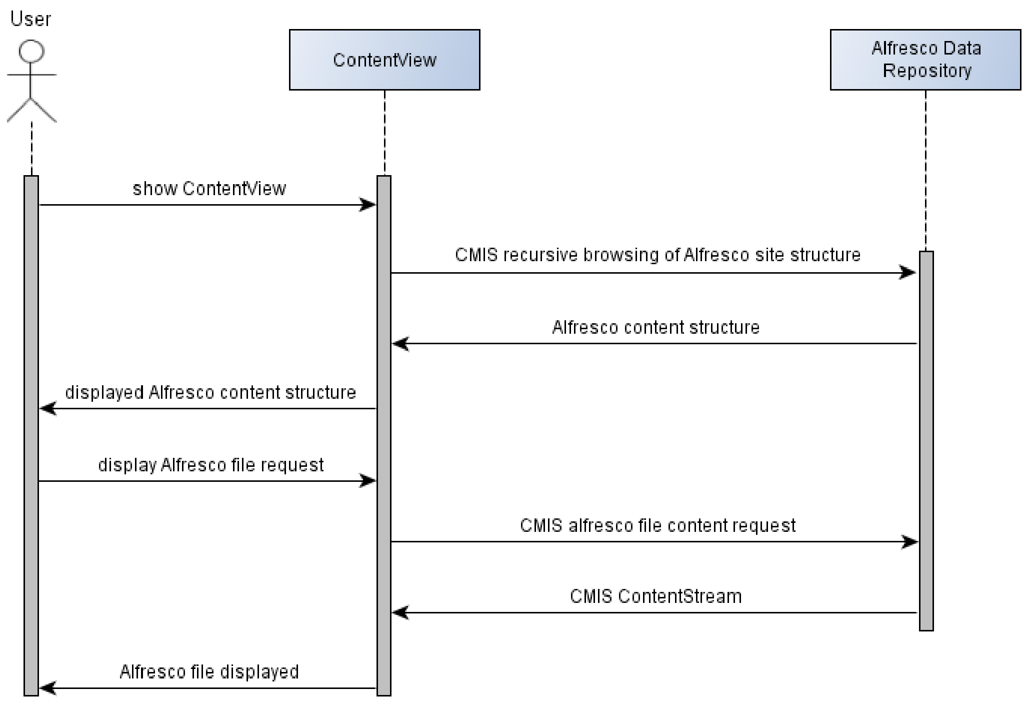

User
ContentView
Alfresco Data Repository
show ContentView
CMIS recursive browsing of Alfresco site structure
Alfresco content structure
displayed Alfresco content structure
display Alfresco file request
CMIS alfresco file content request
CMIS ContentStream
Alfresco file displayed

Figure 11: Sequence diagram of basic principle of displaying of Alfresco.

ContentView is a visualisation component of the Content Repository Client, which enables displaying the Alfresco repository structure in the Eclipse environment. Figure 11 shows a sequence diagram how user access remote content via *ContentView*. All retrieved content is locally cached for further usage in *document* folder.

CMIS defines a data model, which encapsulates the core concepts found in most content repositories. Alfresco provides an implementation of the CMIS bindings and maps the Alfresco content meta-model to the CMIS domain model. This allows content models defined in Alfresco to be exposed and manipulated via CMIS protocol (Alfresco Help, 2013). Figure 12 shows CMIS domain model where main concepts used by content repository client are described here:

- **Document** is similar to a file, it has properties to hold document metadata, such as the document author and modification date and custom properties. It can also contain a content stream.
- **Folder** is self-explained, it is container for other objects. Note, that apart from default hierarchical structure, CMIS is optionally able to store objects in multiple folders or in no folders at all.

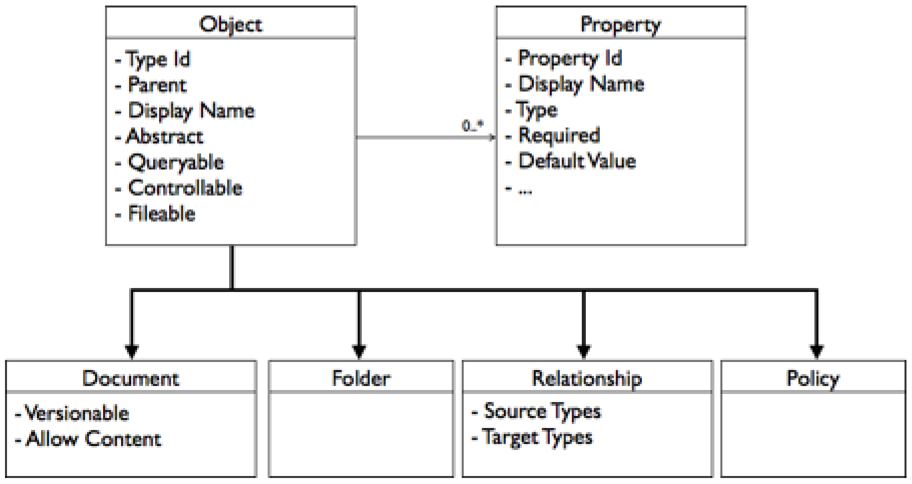


Figure 12: CMIS domain model.

4. SIMULATION ANALYSIS TOOL

4.1. ARCHITECTURE OVERVIEW

The scheme depicted in Figure 13 presents the main plug-ins of the Simulation Analysis Tool.

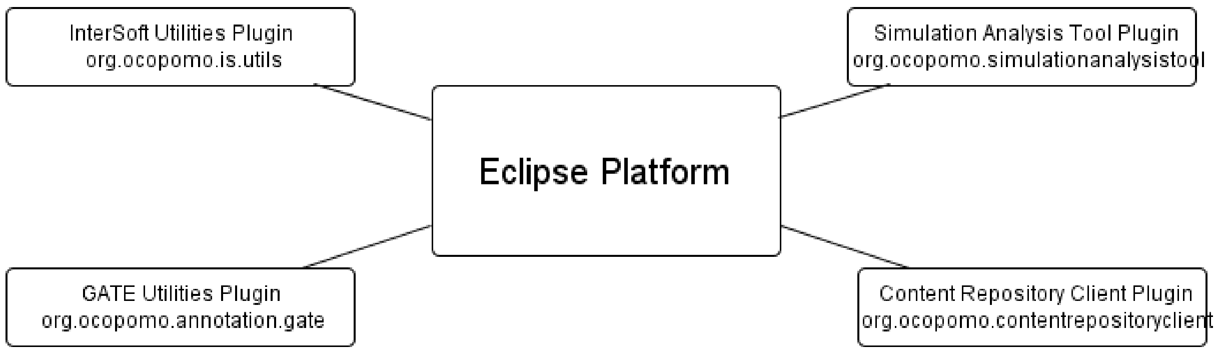

InterSoft Utilities Plugin
org.ocopomo.is.utils
Simulation Analysis Tool Plugin
org.ocopomo.simulationanalysistool
Eclipse Platform
GATE Utilities Plugin
org.ocopomo.annotation.gate
Content Repository Client Plugin
org.ocopomo.contentrepositoryclient

Figure 13: Overview of Simulation Analysis Tool plug-ins.

Following, the main plug-ins are shortly described.

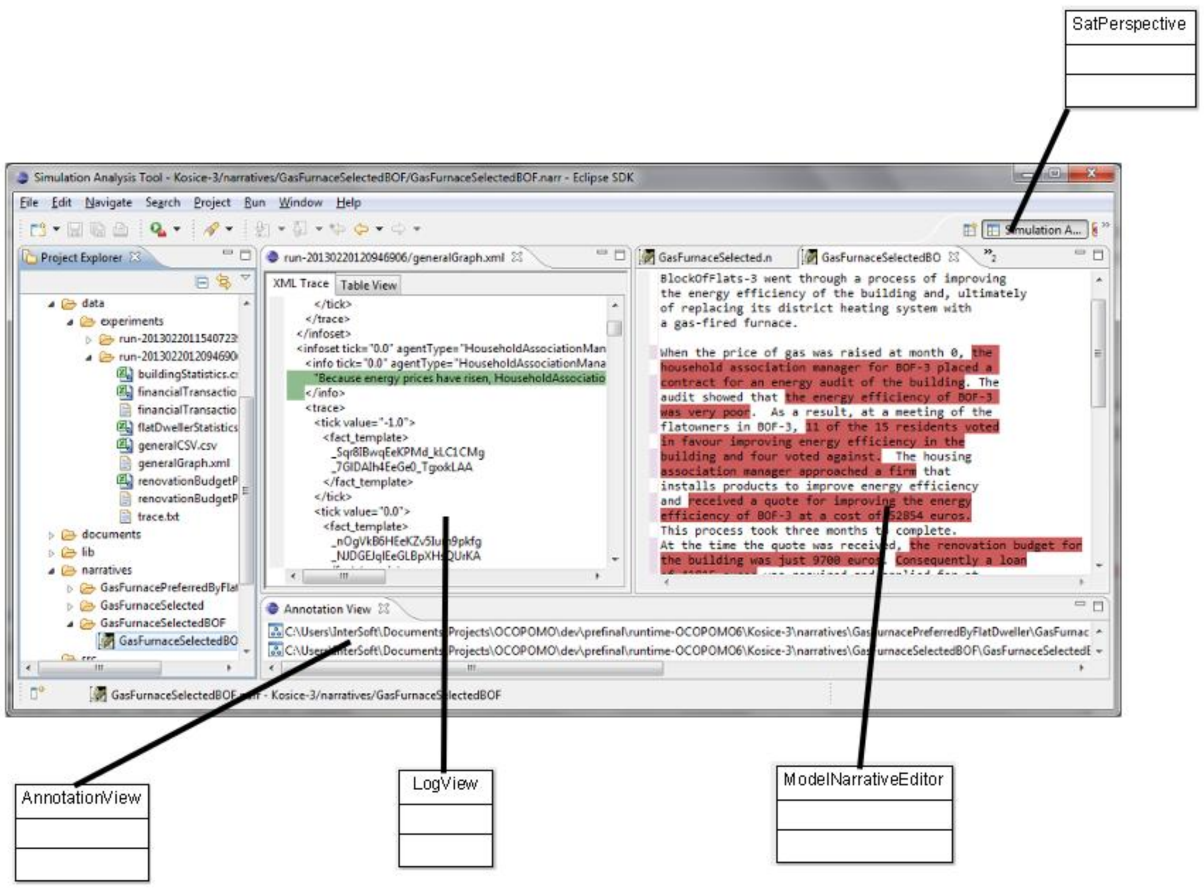
- Simulation Analysis Tool Plug-in (`org.ocopomo.simulationanalysistool`): Plug-in that provides tool for creating and annotating of model-based narrative scenarios.
- InterSoft Utilities Plug-in (`org.ocopomo.is.utils`): Plug-in that provides utilities for parsing specific XML files and logging support.
- GATE Utilities Plug-in (`org.ocopomo.annotation.gate`): Plug-in that wraps GATE core library and provides API for initialization and usage of GATE annotation management.
- Content Repository Client Plug-in (`org.ocopomo.contentrepositoryclient`): Plug-in that provides API and GUI for remote accessing of Alfresco content repository via CMIS and REST protocols.

4.2. SETUP OF DEVELOPMENT ENVIRONMENT AND DEVELOPMENT CYCLE

To start developing new components or to customize Simulation Analysis Tool, the developer should install an Eclipse Indigo version. The developer should further install PLUGIN Development tools and CCD Tools plug-ins from the OCOPOMO and Eclipse Project update sites. It is further necessary to include the Simulation Analysis Tool and Content Repository Client packages into the workspace.

4.3. SIMULATION ANALYSIS TOOL - PLUG-IN DESCRIPTION

We have used basic plug-in mechanism of Eclipse to implement Simulation Analysis Tool. Whole tool is embedded into `org.ocopomo.simulationanalysistool` plug-in. This plug-in contains main implementation of the tool. Figure 14 shows the Simulation Analysis Tool perspective in Eclipse presenting the main visible components of this tool.


SatPerspective
Simulation Analysis Tool - Kosice-3/narratives/GasFurnaceSelectedBOF/GasFurnaceSelectedBOF.narr - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Project Explorer
run-20130220120946906/generalGraph.xml
XML Trace
Table View
GasFurnaceSelected.n
GasFurnaceSelectedBO
Annotation View
AnnotationView
LogView
ModelNarrativeEditor

```
    </tick>
  </trace>
</infoset>
<infoset tick="0.0" agentType="HouseholdAssociationMan
  <info tick="0.0" agentType="HouseholdAssociationMana
    "Because energy prices have risen, HouseholdAssociatio
  </info>
  <trace>
    <tick value="-1.0">
      <fact_template>
        _Sqr8IBwqEeKPMd_kLC1CMg
        _7GIDAIh4EeGe0_TgokLAA
      </fact_template>
    </tick>
    <tick value="0.0">
      <fact_template>
        _nOgVkB6HEeKZv5Iun9pkfg
        _NJDGEJqIEeGLBpXHsQUrKA
```

```
BlockOfFlats-3 went through a process of improving
the energy efficiency of the building and, ultimately
of replacing its district heating system with
a gas-fired furnace.

When the price of gas was raised at month 0, the
household association manager for BOF-3 placed a
contract for an energy audit of the building. The
audit showed that the energy efficiency of BOF-3
was very poor.  As a result, at a meeting of the
flatowners in BOF-3, 11 of the 15 residents voted
in favour improving energy efficiency in the
building and four voted against.  The housing
association manager approached a firm that
installs products to improve energy efficiency
and received a quote for improving the energy
efficiency of BOF-3 at a cost of 52854 euros.
This process took three months to complete.
At the time the quote was received, the renovation budget for
the building was just 9700 euros. Consequently a loan
```

```
C:\Users\InterSoft\Documents\Projects\OCOPOMO\dev\prefinal\runtime-OCOPOMO6\Kosice-3\narratives\GasFurnacePreferredByFlatDweller\GasFurnac
C:\Users\InterSoft\Documents\Projects\OCOPOMO\dev\prefinal\runtime-OCOPOMO6\Kosice-3\narratives\GasFurnaceSelectedBOF\GasFurnaceSelectedE
```

Figure 14: Visible components of Simulation Analysis Tool.

- *SatPerspective (*`org.ocopomo.simulationanalysistool.perspective.SatPerspective`*):* Defines layout of the Simulation Analysis Tool.
- *LogView (*`org.ocopomo.simulationanalysistool.viewer.LogView`*):* Provide viewer for displaying plain text viewer for logs and table viewer for search functionality
- *AnnotationView (*`org.ocopomo.simulationanalysistool.viewer.AnnotationView`*):* Simple table viewer for displaying of all annotations in current edited narrative. Main component for implementation of this viewer is `org.eclipse.jface.viewers.TableViewer`.
- *ModelNarrativeEditor (*`org.ocopomo.simulationanalysistool.editors.ModelNarrativeEditor`*):* Editor used for creation and annotation of model-based narrative scenarios.

The ModelNarrativeEditor and LogViewer modules are described in next sections.

4.3.1. Model Narrative Editor

Model Narrative Editor allows creation of model-based narrative scenarios in plain text format and user can annotate them to related log files. These plain text files could be modified during annotation process. Annotated files are displayed as plain text but they are XML files containing tags that reference text parts of specific log files. XML files are created and managed by GATE library. Displaying of plain text is managed by `org.eclipse.jface.text.source.projection.ProjectionViewer` which allows to hide Master XML Document from users and display only its projection (see Figure 15). Annotated files contains two main parts: textual content mixed with node elements and part of annotation with references to start and end node of textual content with additional information.

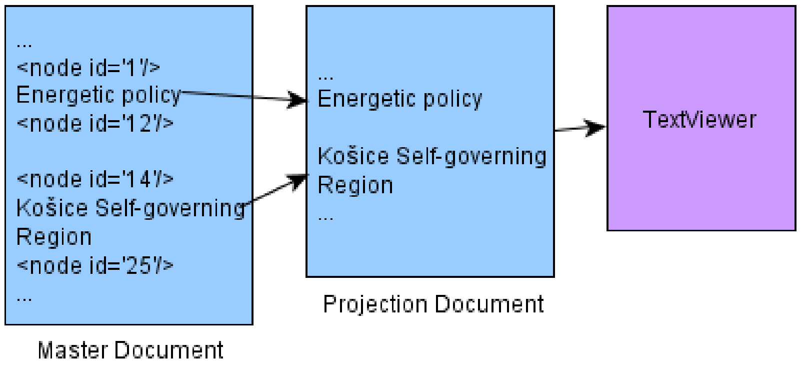


Figure 15: Basic principle of ProjectionViewer.

4.3.2. Log View

LogView uses two components for displaying a content of simulation logs:

- `org.eclipse.jface.text.source.SourceViewer`: This configured viewer is able to display log files in plain text form with its annotations contained in opened model-based narrative scenario. This component does not process displayed content it displays it as is stored in log file.
- `org.eclipse.jface.viewers.TableViewer`: This component parse XML log files, filter all markups and display only relevant formatted content. StAX (http://stax.codehaus.org/) parser is used to parse XML log files because it does not store whole model in memory (it is necessary because log files could be tens of MB length).

4.4. TRACEABILITY EXPLANATION

The basic traceability annotation mechanism of linking a model-based scenario to the respective input materials, i.e. the evidence-based scenarios and background documents, via simulation log records and CCD model elements, is depicted in Figure 16. The text highlighted in the model-based scenario wiki page corresponds to the "simulation-based fact X" since it was derived from the respective simulation log record (i.e., an expert marked this relationship in the Simulation Analysis Tool). The log record refers to one or more CCD model concepts, which were used as annotation marks in evidence-based scenarios and/or background documents.

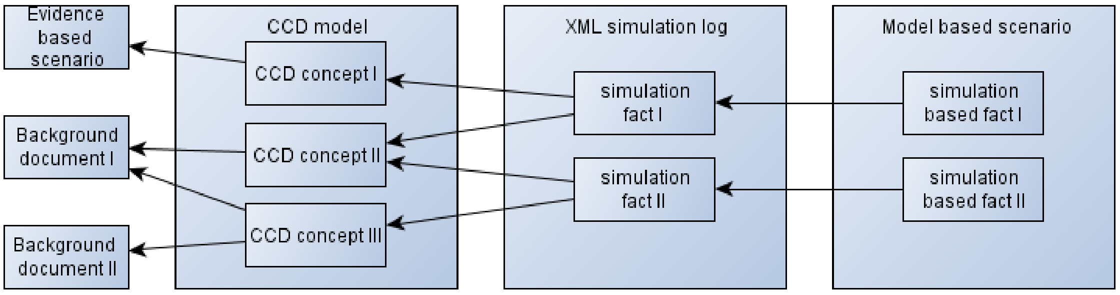


Figure 16: Generating traceability information from annotated model based scenarios.

REFERENCES

Alfresco Help - Alfresco content models and CMIS. Alfresco Software Inc., 2013. Available at http://docs.alfresco.com/3.4/index.jsp?topic=%2Fcom.alfresco.Enterprise_3_4_0.doc%2Fconcepts%2Fcontent-model-cmis.html (last accessed 18 April 2013).

Bird, S. and Liberman, M.: *A Formal Framework for Linguistic Annotation*. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania, 1999.

Cunningham, H. et al: *Text Processing with GATE (Version 6)*. University of Sheffield, Department of Computer Science. 15 April 2011. Available at http://gate.ac.uk/releases/gate-6.1-build3913-ALL/tao.pdf (last accessed 18 April 2013).

Eclipse Platform Plug-in Developer Guide. The Eclipse Foundation, 2005. Available at http://archive.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/org.eclipse.platform.doc.isv.3.1.pdf.zip (last accessed: April 18th, 2013).

Fielding, R. T.: *Architectural Styles and the Design of Network-Based Software Architectures*. Ph.D. Dissertation. University of California, Irvine, 2000.

Grishman, R: *TIPSTER Architecture Design Document*, Version 2.3. Technical report, DARPA, 1997. Available at http://www.itl.nist.gov/div894/894.02/related_projects/tipster/ (last accessed 18 April 2013).

Ide, N., Bonhomme, P., and Romary, L.: *XCES: An XML-based Standard for Linguistic Corpora*. In: Proceedings of the Second International Language Resources and Evaluation Conference (LREC), pp 825-830, Athens, Greece, 2000.

OMG MOF - Meta Object Facility 2.0 Query/View/Transformation Specifcation, v1.1. OMG Available Specification, Document Number: formal/2011-01-01 (2011, January 1). Technical report. Object Management Group, 2011.

OMG MOFM2T - MOF Model to Text Transformation Language, v1.0. OMG Available Specification, Document Number: formal/2008-01-16 (2008, January 16). Technical report. Object Management Group, 2008.

Scherer, S., Wimmer, M. A., Moss, S., Smatana, P., and Furdik, K.: *D4.2-B: User Manual on CCD Tools*. Annex to Deliverable 4.2, OCOPOMO consortium, 2013.

Smatana, P. and Furdik, K.: *D4.2-D: User Manual on Tools for Simulation Analysis and Output Scenario Generation*. Annex to Deliverable 4.2, OCOPOMO consortium, 2013.

Steinberg, D.: *Eclipse modeling framework*. The Eclipse series, 2. ed. Upper Saddle River, NJ, Addison-Wesley, 2009.