Topic Maps for the Cambridge Advanced Modeller

by

Clive Stevens (CTH)

Fourth-year undergraduate project

in Group C, 2011/2012

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: _____________________. Date: ____________

Topic Maps for the Cambridge Advanced Modeller

by Clive Stevens (CTH)

Fourth-year undergraduate project in Group C, 2011/2012

Technical abstract

This project aims to create a Topic Maps toolbox for the Cambridge Advanced Modeller (CAM). The toolbox should allow creation of simple Topic maps and strict Topic maps, and should also provide users with the ability to perform suitable Topic Maps operations. The motivation behind this project is to open up Topic Maps technology to University of Cambridge Engineering Design Centre (EDC) researchers and collaborators, a group of people already familiar with CAM.

Topic Maps is a standard for information management and interchange. A Topic map represents a collection of statements, in which attributes (termed Occurrences) and interrelations (Associations) are specified for primary subjects (Topics). A Topic can represent a subject as abstract as the user desires – anything from a physical object, to an idea, to an opinion on an idea. Topic Maps is a concept formally defined in ISO 13250 (referred to in this project as strict Topic Maps), making it suitable for computer implementation.

Simple Topic Maps is a concept defined in the context of this project to be a simplified version of the strict Topic Maps standard. Defining fewer constructs, it is easier to use, but restricts possible analysis.

CAM is a configurable software platform developed by the EDC, which allows users to model, simulate, and analyse processes (typically design processes of products, or service processes). Models consist of instance elements joined by instance connections, both of which may have a number of instance properties. Instance properties are used to specify attributes of instance elements and instance connections, for example, an instance element may have a Name instance property. CAM defines a modular interface, which permits toolbox development and makes this project possible. Toolboxes typically define new model types.

A Topic Maps toolbox has successfully been developed. Any Topic map conforming to the strict Topic Maps standard may be represented in full using the toolbox. Simple Topic maps may also be created using this toolbox.

The Topic Maps toolbox includes a number of new property interface implementations that

define the new types of instance property required for Topic Maps. These include a property allowing the selection of a single instance element from a model, a property allowing a list of items to be created, a property with an editable drop-down box user interface component, and a property with a tree user interface component. These property interface implementations have been developed in a generic way, and may be useful to future toolboxes.

Various Topic Maps operations are provided by the toolbox – version conversion, validation, and export operations. The version conversion operations convert strict Topic maps into simple Topic maps and vice versa. The validation operation and export operation are only defined for strict Topic maps. The validation operation checks if a Topic map conforms to the Topic Maps standard. The export operation exports a Topic map to an XTM document that may be used in other Topic Maps applications.

There are several Topic Maps operations that remain to be implemented. An import operation and a merge operation will be developed before the toolbox is made available from the CAM website on 1st July 2012. The import operation is essentially the reverse of export – a CAM Topic map model is created from a Topic map defined in an XTM document. The merge operation detects and merges any duplicate items in a Topic map.

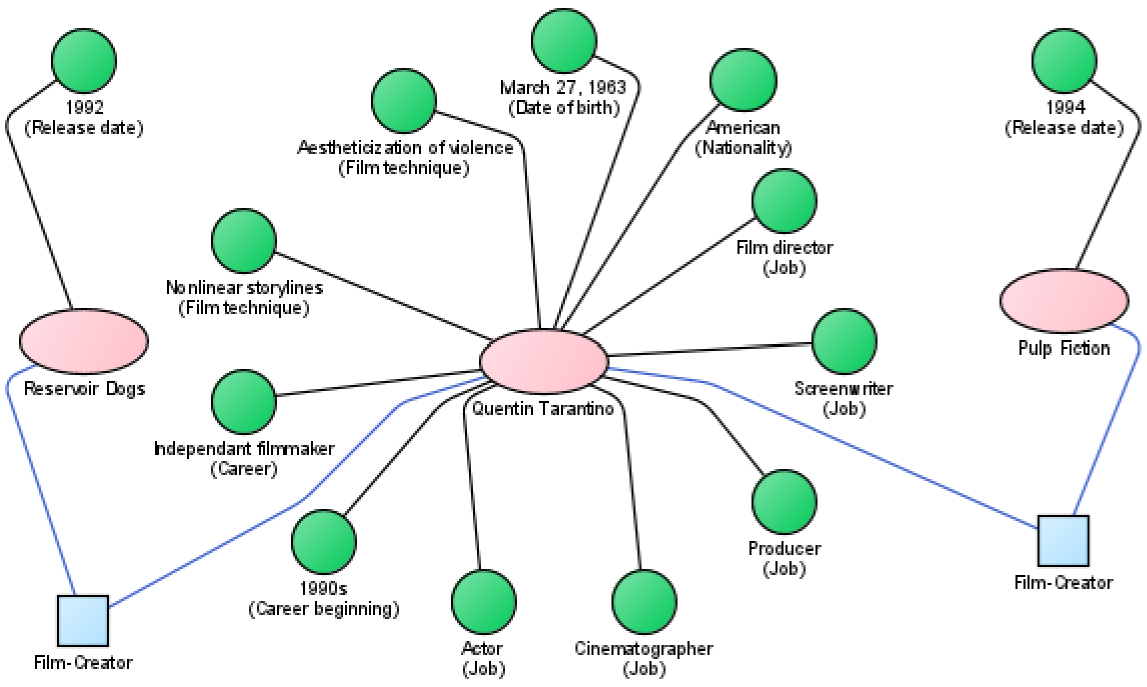


Figure 1: *A Topic map created using the CAM Topic Maps toolbox to discuss Quentin Tarantino. Topics are represented by pink circles, Associations by blue squares, and Occurrences by green circles.*

Contents

Acknowledgements

I would like to thank both of my supervisors for the assistance they have given me with this project – Dr Nicholas Caldwell was particularly helpful in discussions regarding the direction of my project, and Ms Seena Nair proved invaluable when discussing technical issues.

I would also like to thank H. Nam Le, an EDC diploma student, for conducting a brief usability test on the Topic Maps toolbox.

Notation

Topic Maps refers to the technology, Topic map refers to a Topic map item.

1. Introduction

The solution-neutral problem statement of this project was to enable researchers and collaborators of the Engineering Design Centre, University of Cambridge (the EDC) to capture information.

The following critical decisions were made prior to finalising the project objective:

- Information would be captured in the form of a Topic map. Topic Maps is a technology specifically designed for this purpose.
- Topic maps would be created in the Cambridge Advanced Modeller (CAM) for reasons discussed below.

The final objective of this project, therefore, was:

> *"To design and implement Topic map representation and algorithms for use in the Cambridge Advanced Modeller software platform"*

Topic Maps is a technology for managing knowledge and information. This is an area of particular interest to the EDC. It is, therefore, useful to add Topic Maps functionality to CAM, a stable application already familiar to EDC researchers.

The EDC are attempting to consolidate the tools that have been developed on the CAM platform, enabling their researchers to easily explore links between related models through the same piece of software. Adding a Topic Maps toolbox will allow a greater number of links to be explored.

Some research partners of the EDC are heavily involved in Topic Maps. Adding a Topic Maps extension to CAM will allow the EDC to work more closely with their collaborators inside the CAM platform, creating more research opportunities.

This project's objective is not to create a revolutionary new Topic Maps application, but to open up Topic Maps technology to a specific group of people. The Topic Maps application resulting from this project is, therefore, a lot simpler and less functional than the visual Topic Maps editor proposed by Thorsten Witt [1]. His proposal is for a complete, standalone Topic Maps application with similar structure to the Ontopia suite and Networked Planet Web3 platform (discussed in section 1.1.5), with the addition of a complex visual Topic map editor.

The remainder of this chapter introduces Topic Maps as a method of representing information, and gives an overview of the CAM software platform.

1.1. Topic Maps

Topic Maps is a technology that allows information to be formally recorded, viewed and analysed. Developed in the 1990s, it is a relatively mature technology that has been standardised in ISO 13250 [2].

1.1.1. Theory: information

All information, even that without structure, can be viewed as a collection of statements discussing a number of subjects. The wording of the statements and the choice of subjects are not unique, however, and must be decided after careful consideration. For example, consider the following information on Quentin Tarantino (an extract from Quentin Tarantino's Wikipedia entry [3]):

> Quentin Jerome Tarantino [4] (… born March 27, 1963) is an American film director, screenwriter, producer, cinematographer and actor. In the early 1990s, he began his career as an independent filmmaker with films employing nonlinear storylines and the aestheticization of violence. His films include Reservoir Dogs (1992), *[and]* Pulp Fiction (1994) …

The following statements have been extracted from this information. Sensible subjects have been italicised (and square bracketed where required for clarity), other possible subjects have been underlined.

1. According to *[Filmreference.com - Quentin Tarantino Biography (1963-) [4]], Quentin Tarantino*'s full name is *Quentin Jerome Tarantino.*
2. *Quentin Tarantino* was born on *March 27, 1963*.
3. *Quentin Tarantino*'s nationality is *American*.
4. *Quentin Tarantino* works as a *[film director]*, *[screenwriter]*, *[producer]*, *[cinematographer]* and *[actor]*.
5. *Quentin Tarantino*'s career began at some point in the *[decade beginning in 1990]*.
6. *Quentin Tarantino*'s career is an *independent filmmaker*.
7. *Quentin Tarantino* uses the following techniques in his films: *nonlinear storylines* and the *aestheticization of violence.*
8. *Reservoir Dogs* is a film by *Quentin Tarantino*.
9. *Pulp Fiction* is a film by *Quentin Tarantino*.
10. *Reservoir Dogs* was released in *1992.*

11. *Pulp Fiction* was released in *1994*.

The subjects that have not been identified as sensible are better interpreted as the nature of relationships between sensible subjects. For example, in statement 3, the relation between the subjects Quentin Tarantino and American is nationality.

Considering the subjects from all statements other than 1, 8 and 9[1], two distinct categories of subject are apparent. Some subjects appear in multiple statements and are subjects that more statements may be made about (e.g. Quentin Tarantino), whereas some subjects appear just once and are only of interest in that particular statement (e.g. American). This is the distinction between data and attribute (data about data) often noted in Computer Science (for example, XML uses Attributes to represent data about data, and Elements to represent data).

Subjects are classified as attributes if they are only of interest in the context of the statement they appear in. This classification is not unique, however, and may even change with purpose. Table 1.1 shows a possible, sensible classification.

Data items	**Attributes**
Quentin Tarantino	March 27, 1963
Reservoir Dogs	American
Pulp Fiction	Film director
	Screenwriter
	Producer
	Cinematographer
	Actor
	Decade beginning in 1990
	Independent filmmaker
	Nonlinear storylines
	The aestheticization of violence
	1992
	1994

Table 1.1: *A classification of subjects from statements 2-7 and 10-11 of the Tarantino example.*

Statements 8 and 9 define the relations between multiple data items in a similar way to the data-attribute statements 2-7 and 10-11. This distinction is important to note as data-attribute statements are represented differently to data-data statements in Topic Maps.

1 Statements 1, 8 and 9 follow a different structure and will be discussed next

Statement 1 can be thought of as giving an alternate name to a subject. Quentin Tarantino and Quentin Jerome Tarantino refer to the same subject with different names.

1.1.2. Topic Maps constructs

A Topic map can be viewed as a network of connected items. Each Topic Maps item is an instance of a Topic Maps construct, which can be thought of as a schema for the item. The different constructs are:

- Topic map – The whole network of items is represented by a Topic map item.
- Topic – A Topic item represents a single data item.
- Topic name – Each Topic item may have zero or more Topic name items, representing subject names for the Topic.
- Variant – Each Topic name item may have zero or more Variant items, variations of the Topic name.
- Occurrence – Each Topic may have zero or more Occurrences, each representing a single attribute.
- Association – A data-data relation (a relation between multiple Topics) is represented by an Association item.
- Association role – Each Association may involve multiple Topics, each of which have a specific reason for being in the Association. These reasons are represented by Association role items.

Example

Consider the Tarantino example. A Topic map might be created to represent the entire collection of statements (Figure 1.1). All subjects classified as data in Table 1.1 would be represented by Topics, and all subjects classified as attributes would be represented by Occurrences. The Topic representing Tarantino would have two Topic names – *Quentin Tarantino* and *Quentin Jerome Tarantino*. Statements 8 and 9 would each be represented by an Association. The Association representing statement 8 would have two Association roles, representing the roles played by the Tarantino Topic and the Reservoir Dogs Topic in the Association.

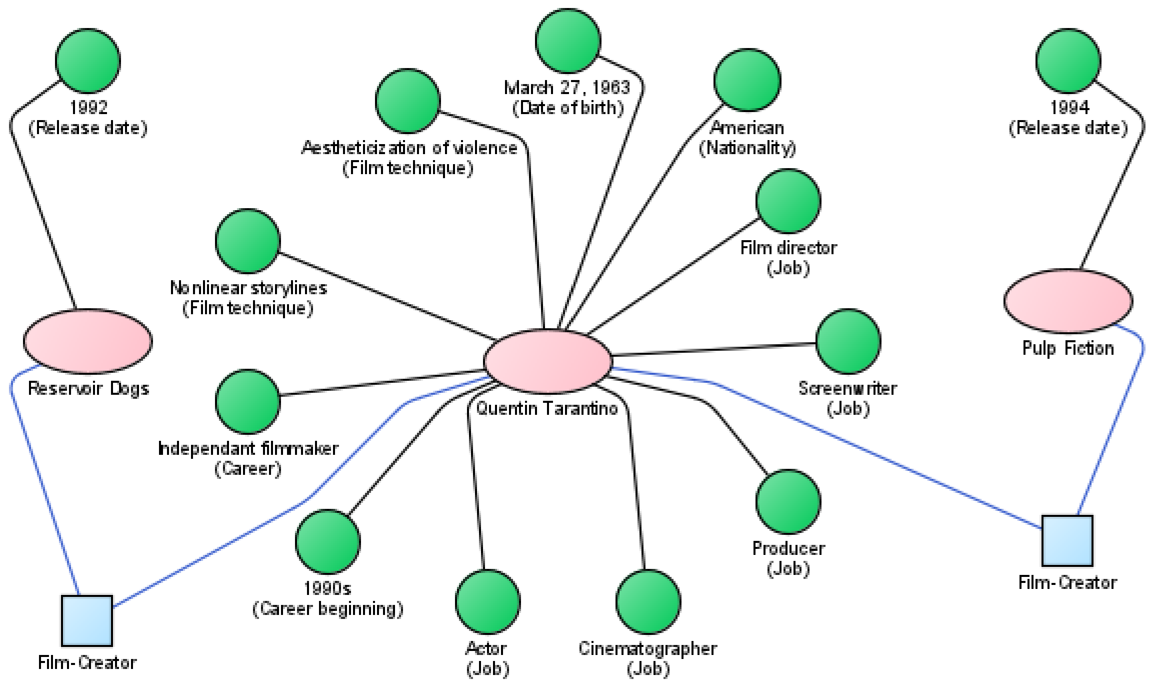


Figure 1.1: *A Topic map representation of the Tarantino example (created in CAM). Pink ovals represent Topic items, green circles Occurrence items, blue squares Association items, and blue connections Association role items.*

Further discussion

A Topic may represent any conceivable subject, from a physical object, to an idea, to an opinion on an idea. There is an infinite number of possible subjects, which makes identifying individual subjects difficult. This is essential, however, as a statement is only meaningful if the subjects it refers to are known. A subject indicator is defined as the smallest amount of information that unambiguously identifies a particular subject. A Topic is linked to its subject by either the address of the subject itself (through a Subject locators property), or the address of a subject indicator defining the subject (through a Subject identifiers property). Topics may have many subject identifiers and subject locators, each of which individually attempts to identify the subject.

Topic names and Variants do not attempt to identify the subject of Topic items, but facilitate discussion and analysis of Topic maps. Both Variants and Topic names have a Value property, which is set to the subject name the item represents. The distinction between Topic names and Variants is somewhat hazy, but generally a subject name should be classified as a Variant if it is simply a translation of an existing Topic name to another notation or language. Consider a

Topic representing the Russian city Saint Petersberg. This Topic may have three Topic names: the city's current name (Saint Petersberg), and its two past names (Petrograd and Leningrad). The Saint Petersberg Topic name may have many Variants: it is referred to as Petersberg in Russian literature [5], colloquially referred to as Peter [5], and may be written in short-hand as St. Petersberg. Each Topic name has a specific nature, which is defined in a Type property. Variants do not have this property as their type is implicitly defined as being equal to their parent Topic name's type.

Occurrences have a Type property, defining the nature of the relation from the Occurrence to its Topic. Associations and Association roles also have Type properties, defining the nature of the Association, and the nature of the role played, respectively. Considering the Tarantino example, the Occurrence American would have the Type Nationality. Statement 8 would require the Topic Quentin Tarantino to play the Association role of Type Creator, and Reservoir Dogs to play the Association role of Type Film, in an Association of Type Film-Creator.

1.1.3. Topic Maps properties

There are a number of other Topic Maps properties, in addition to those mentioned in section 1.1.2. All Topic Maps properties are discussed below (boldface text is the name of a property).

Subject identifiers and **Subject locators** are properties of Topics and are discussed in section 1.1.2.

Value is a property of Topic names, Variants and Occurrences. It is discussed in section 1.1.2.

Type is a Topic defining the nature of Topic names, Occurrences, Associations and Association roles. It is discussed in section 1.1.2.

Item identifiers is a property of all Topic Maps items. An item identifier uniquely identifies the item it is a part of.

Scope is a property of Topic names, Variants, Occurrences and Associations. It is a collection of Topic items that states when the particular item is valid.

Datatype is a string specifying how the Value property should be interpreted. Variants and Occurrences have datatype properties. Topic names have the fixed datatype: string.

A Topic may be created to further discuss a particular item in a Topic map. In other words, a Topic may be created whose subject is another item in the Topic map. This process is known as reification. All Topic Maps constructs, apart from Topics, have a **reifier** property that may identify a Topic whose subject is the item in question.

The relations between all Topic Maps constructs are also defined as properties. Topic maps have the properties **Topics** and **Associations**. Topics have the properties **Topic names** and **Occurrences**. Topic names have the property **Variants**. Associations have the property **Association roles**. Association roles have the property **Player** (linking to the Topic that plays the particular Association role in the Association).

Consultation of the Topic Maps Datamodel ISO standard [2] is recommended for a more detailed discussion of Topic Maps constructs and their properties.

1.1.4. Simple Topic Maps

In some situations, a Topic map conforming to the Topic Maps standard may be too complex. The user may want to create a simple Topic map without concerning themselves over the finer points of the standard. Therefore, this project introduces the concept of simple Topic Maps, a simplified version of the full Topic Maps standard.

No formal specification for simple Topic Maps exists. In the context of this project, a simple Topic map will consist of the four main Topic Maps constructs – Topics, Occurrences, Associations and Association roles – each with a reduced number of properties. Topics will have two properties – Subject, a description of the subject, and Occurrences, the Occurrences belonging to the Topic. Occurrences will have two properties – Value, the attribute represented by the Occurrence, and Type, a Topic item equal to the Type of the Occurrence. Associations will have a single property – Association, a description of the Association type. Association roles will also have a single property – Role played, a description of the role played by the Topic in the Association.

1.1.5. Current Topic Maps applications

Ontopia is a mature suite of open source Topic Maps applications containing a HTML form based editor, a graphical visualiser (displays Topic maps as a collection of connected nodes, Figure 1.2), and a browser (displays Topic maps as a set of linked web pages) [6]. All user actions are made through a HTML interface to a core engine that sits behind an Apache Tomcat server.

Networked Planet have a product called Web3 Platform. While not marketed as such, it can be identified as a Topic Maps application from the introductory video [7] (it is marketed as a solution for managing and publishing semantic data). This video shows a visual Topic Maps constraint editor[2] and a HTML form based Topic map editor (Figure 1.3). All user actions are made through an Internet Information Services (IIS) server to a SQL server.

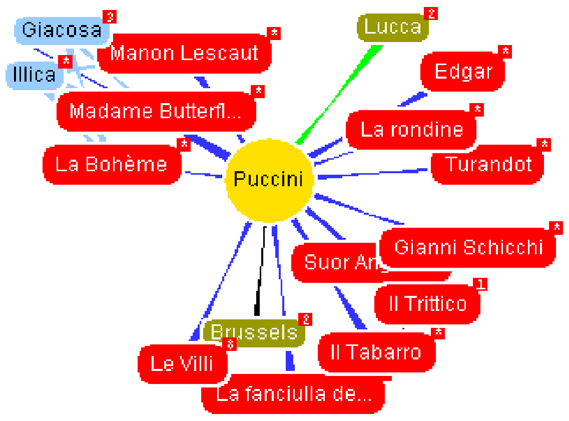
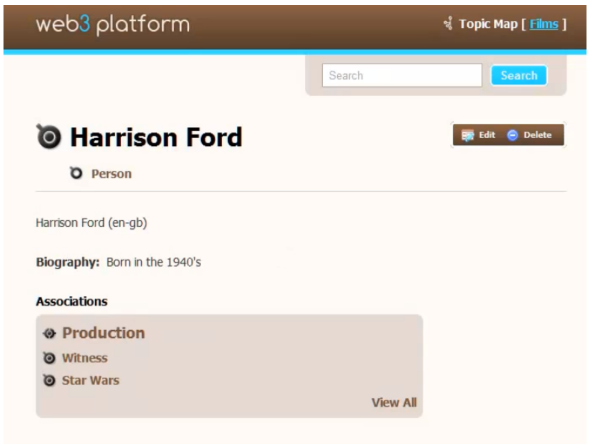


Figure 1.2: *Screenshot of Vizigator, the Ontopia graphical visualiser (from the Ontopia website [8]).*



Harrison Ford

Person

Harrison Ford (en-gb)

Biography: Born in the 1940's

Associations

Production

Witness

Star Wars

Figure 1.3: *Screenshot of the Networked Planet Web3 platform (from the Web3 introductory video [7]).*

1.2. Cambridge Advanced Modeller

CAM is a Java software platform developed and maintained by the EDC. It includes facilities for creating and visualising many types of linkage model using diagrams, Dependency Structure Matrices (DSMs) or force-directed network layouts [9]. It is free for research, teaching and evaluation use [10].

1.2.1. Beginnings

CAM was originally named P3 Platform, and was developed to facilitate modelling, simulation and analysis of the dependencies and flows in complex systems. Typical complex systems would be design processes of products, or service processes, where simulation and analysis could be used to help identify bottlenecks, optimise processes and identify unnecessary rework. Specifically, the P3 Platform allowed users to create and analyse Applied Signposting Models (ASMs), which can be thought of as flow charts with underlying logic that gets executed when analysis is performed.

2 A Topic Maps constraint document can be thought of as a schema for a Topic map

1.2.2. Diversification

As the P3 Platform evolved, it was noticed that modifications could be made to extend its application. The existing user interface could be retained, and a modular interface implemented, allowing toolboxes to be created to support other model types. These modifications were done, and the project renamed to CAM, an extensible platform that would allow users to create projects involving multiple models of different types.

Since then, a number of toolboxes have been created [11], such as an ASM toolbox (adapted from the original P3 Platform ASM implementation), a DSM toolbox, and a Change Prediction Method toolbox. While many toolboxes have already been developed, opportunities for new toolboxes are often identified, such as the Topic Maps toolbox developed in this project.

1.2.3. Use

In CAM, projects are viewed as workspaces containing multiple workbooks, and each project is stored in a single XML file. Each workbook is a different model with a specific model type. Workbooks may be sub-divided into worksheets at the user's discretion to logically sub-divide the model.

Models are created by dragging and dropping schema element icons (Figure 1.4f-j) from a palette onto a worksheet. Instance connections can be made between instance elements (Figure 1.5) using the connection tool (Figure 1.4c). The schema elements on the palette, the way they look and their properties are all defined by the toolbox in current use. The possible instance connections, and their properties are also defined in the toolbox.

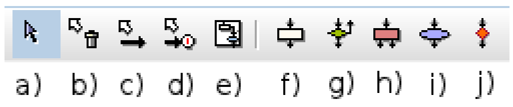


Figure 1.4: *The ASM CAM palette. Icons a-e are defined by CAM and f-j are defined by the ASM toolbox.*

The possible simulation and analysis is defined at a toolbox level, and can be carried out by selecting appropriate options from the Tools and Analysis menus.

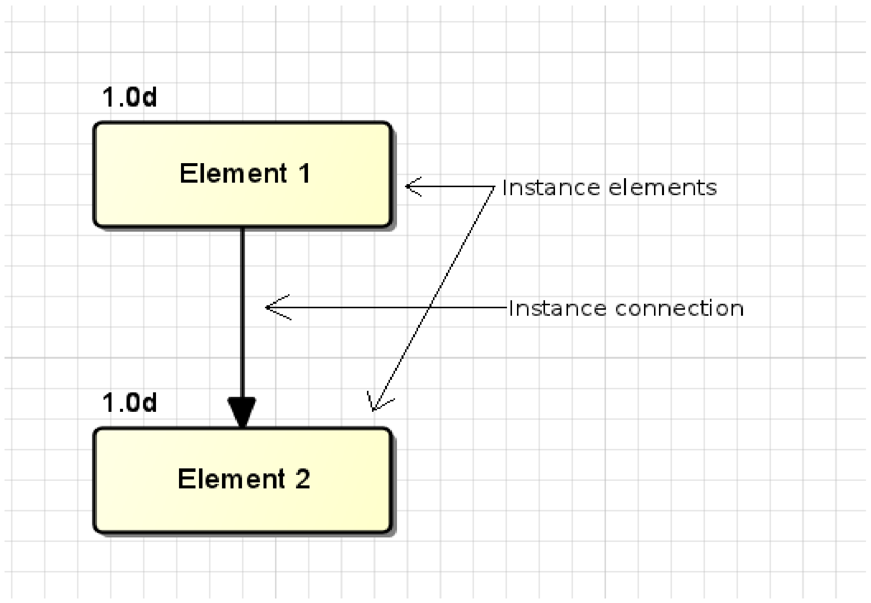


Figure 1.5: *A small ASM model in CAM.*

2. Requirements and analysis

This chapter considers the functional requirements of any strict or simple Topic Maps application, discusses the various CAM implementation details that influence the Topic Maps toolbox requirements, and formally states the implementation requirements for the toolbox.

2.1. Topic Maps requirements

The user must be able to create simple and strict Topic maps. The user may have difficulty deciding on the particular version to choose when starting a new Topic map, and may wish to change versions mid way through creation. Methods for converting between the two versions are, therefore, required.

2.1.1. Strict Topic Maps

A strict Topic Maps application must provide the user with methods to create, modify and delete an item of any Topic Maps construct (section 1.1.2). The modification method should permit the user to change any of item's properties (section 1.1.3).

Various Topic Maps operations exist to aid the analysis and interpretation of a Topic map. For completeness, a Topic Maps application should allow the user to:

- Determine if a particular Topic map is valid according to the Topic Maps standard.
- Use Topic Maps applications interchangeably, by implementing appropriate import and export algorithms.
- Merge a Topic map, an operation where duplicate Topic Maps items are identified and merged into a single item, reducing redundancy [2].

More complex operations also exist for constraining the structure of a Topic map [12], and for querying a Topic map [13]. While useful, these two procedures are particularly complex and will not be implemented in this project.

2.1.2. Simple Topic Maps

To create a simple Topic map, the user must be able to create, modify and delete an item of any of the simple Topic Maps constructs (defined in section 1.1.4). The modification method must allow the user to modify any of the item's properties (also defined in section 1.1.4).

Aside from conversion to a strict Topic map, no operations exist for a simple Topic map.

2.2. CAM structure

As mentioned in the Introduction, CAM has a modular interface, permitting toolbox development. Toolboxes, depending on their complexity, may contain one or both of the following components:

- A CAM module – A Java package containing additional modular interface implementations and classes required by the toolbox. These packages are all located in the *modules* subdirectory of the CAM installation. Toolboxes requiring implementations of one or more of the CAM modular interfaces (section 2.3) have a module component.
- A CAM palette – A set of text files defining the palette's schema elements, and the palette configuration. Each schema element defines the instance properties and appearance of its instance elements. The palette configuration defines the permissible connections between instance elements of different types. Palettes are located in their own subdirectory of the *palette* subdirectory of the CAM installation. Toolboxes implementing a new model type require a CAM palette component.

The contents of the *modules* and *palette* directories are indexed at runtime, and included in the program. This design allows end-users to tweak a palette's default look without needing to modify and rebuild any code, and allows power-users to create new modules without having to rebuild the core CAM modules.

Consider a family tree model type. At the simplest level, this would require the ability to uniquely identify individuals, and define the relationships between them. Assuming an individual's name is unique, a simple CAM palette could be created to define the family tree model type.

2.2.1. Data-model and meta-model

Details

After loading the contents of the *modules* and *palette* directories, CAM creates a network of schema objects (schema elements and schema properties) known as the CAM meta-model.

These schema objects act as factories[1] for instance objects (instance elements and instance properties), which, when created, are added to the CAM data-model (Figure 2.1).

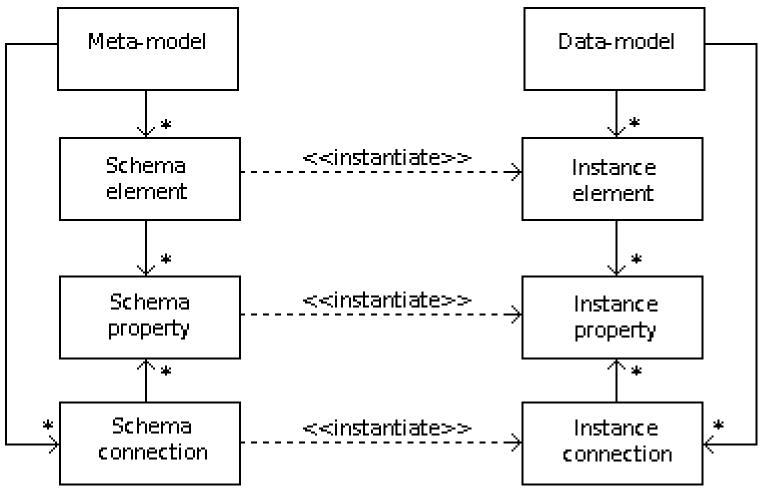


Figure 2.1: *CAM's meta-model and data-model design.*

Graphical representation

Schema elements are represented by icons on the palette toolbar (Figure 1.4). Dragging and dropping a schema element icon from the toolbar onto the worksheet triggers the schema element to instantiate an instance element, displayed at the drop location.

Example

Considering the family tree example, a family tree palette would define one schema element, named *Person*, which would have two schema properties – *Full name* and *Title*. If CAM is executed, schema objects will be instantiated (to represent the *Person* schema element, and *Full name* and *Title* schema properties) and added to the meta-model. If the user triggers a *Person* instance element to be created, it would be given two instance properties, one created by each of the *Full name* and *Title* schema properties. The newly created *Person* instance element would then be added to CAM's data-model (along with its instance properties).

The palette toolbar would look similar to the ASM palette toolbar (Figure 1.4), but would have a single icon representing the *Person* schema element, in place of the five ASM schema element icons.

1 A factory is part of a computer programming design pattern where one object (the factory) is used to instantiate other objects.

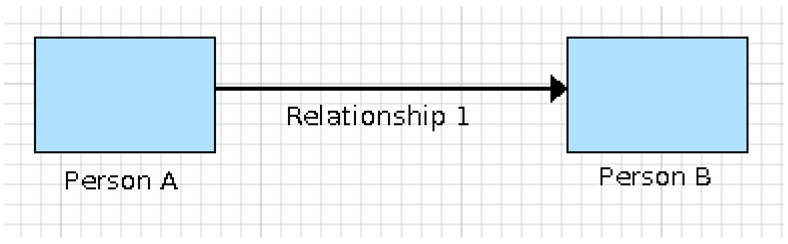


Figure 2.2: *A family tree worksheet, on which are two* Person *instance elements connected by a* Relationship *instance connection.*

2.2.2. Connections

Details

In a similar manner to declaring schema elements, each palette may define one or more schema connections, each of which may have a number of schema properties. Schema connections are added to CAM's meta-model when created, similarly to schema elements. Each palette may also declare constraints on the permitted connections, which each specify a schema connection and two schema elements. Instance connections of the schema connection are then used to connect instance elements of the two schema elements.

When the user creates a connection between two instance elements, CAM determines the schema of the created instance connection from the palette's constraints. An instance connection is instantiated from this schema and added to the CAM data-model.

Graphical representation

Instance connections are created using the connection tool (Figure 1.4c) to join two instance elements. Once created, an instance connection appears as a visual link between the two instance elements.

Example

Considering the family tree example, a family tree palette would define one schema connection, named *Relationship*, which would have one schema property – *Relationship name*. Also defined in the palette would be a single constraint, detailing two *Person* instance elements may be connected using a *Relationship* instance connection.

If the connection tool is used to join two *Person* instance elements, a *Relationship* instance connection would be created and added to the data-model. This instance connection would have a *Relationship name* instance property.

2.3. CAM modular interface

Modular interfaces are plug-in points in CAM's implementation designed to be extended by toolboxes. Two such interfaces are discussed in this section.

2.3.1. Property interface

Users edit instance properties of an instance element or instance connection through a dialog window containing one panel for each property of that element. Changes to instance properties persist to CAM's data-model when the window is shut. Implementations of the property interface define new instance property implementations, specifying the structure of the property's data and how the user edits this data.

Recall the *Person* schema element from the family tree example has two schema properties – *Full name* and *Title*. The *Full name* instance property data is a simple string of text, so an appropriate editor component for this might be a text field. The *Title* instance property data is a string equal to Mr, Mrs, or Miss. While this could be edited using a text field, the finite number of possible values suggest a drop-down box would be more intuitive. A text field property interface implementation with the universal identifier (UID) `p3-single-line-string-property` and a drop-down box property interface implementation with the UID `p3-list-selection-property` exist in one of the core CAM packages (Listing 2.1 and Figure 2.3).

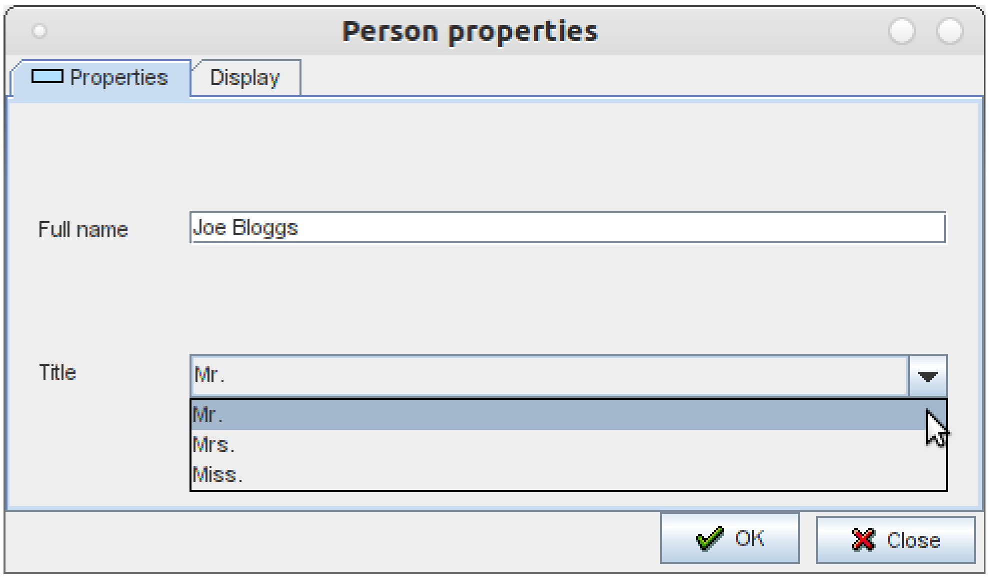
The core CAM packages contains many other property interface implementations, allowing font selection, and colour selection, among others.

```
:class
class-name: "Person"

:properties
property-name: "Full name" "Joe Bloggs"
property-module: p3-single-line-string-property

property-name: "Title"
property-module: p3-list-selection-property
property-initialiser: "<initialiser><option>Mr.</option>
   <option>Mrs.</option><option>Miss.</option></initialiser>"
```

Listing 2.1: *An extract from a CAM palette file defining the* Person *schema element.*



Full name Joe Bloggs

Title Mr.

Figure 2.3: *A properties dialog window for the Person element defined in Listing 2.1.*

2.3.2. Menu item interface

Recognising each model type may require its own tools and analysis, CAM provides an interface to define new menu items, the selection of which, typically, would run a tool or an analysis. When the user switches between workbooks, the contents of the *Tools* and *Analysis* menus update to display menu items relevant to the new workbooks model type[2].

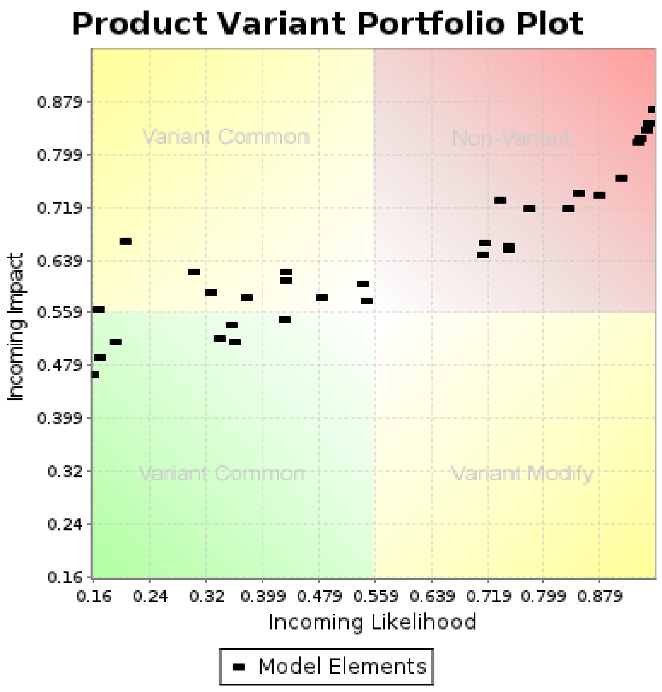


Figure 2.4: *A Product Variant Portfolio Plot of a CPM jet engine model.*

2 At the time of writing, however, CAM has a bug causing new menu items to be retained even if they are not relevant to the new workbook

For example, the Change Prediction Modelling (CPM) toolbox defines several implementations of the menu item interface, used for creating various types of plot. Selecting one of these menu options opens a dialog window, on which the plot options may be modified. After the options have been confirmed, a plot is created and displayed in CAM (Figure 2.4).

2.4. Software requirements specification

2.4.1. Topic map version selection

When the user starts creating a Topic map, they must be able to select the Topic map's version. This functionality was achieved by defining two palettes, one for simple Topic maps and one for strict Topic maps. The Topic map version is specified implicitly by the workbook's palette.

2.4.2. Strict Topic Maps visual instance elements

A Topic Maps construct may be represented in CAM as a schema element or a schema property. The property interface implementations required for this toolbox depend on the palette's configuration – property implementations are likely to be required for the Topic Maps constructs represented by schema properties.

Topic name items and Variant items could be considered as attributes of Topic items, and, therefore, are more appropriately represented by instance properties of Topic instance elements. If Association role items were to be represented by instance elements, they would always have a single connection to a Topic (the Player) and a single connection to an Association (the Association in which the role is played). Association role instance elements, therefore, are redundant, and can be better represented by a single connection.

The resulting palette defines each Topic map item as a single workbook. Topic, Occurrence and Association constructs are defined as schema elements. Schema connections represent the Association role construct (a connection between a Topic and an Association) and the Occurrences property of the Topic construct (a connection between a Topic and an Occurrence). Topic name items and Variant items appear in Topic instance elements' instance properties.

2.4.3. Simple Topic Maps visual instance elements

Because the structure of a simple Topic map is similar to that of a strict Topic map, the two Topic map palettes should appear similar. Therefore, each simple Topic map is represented by a single workbook, the simple Topic, Occurrence and Association constructs are represented by schema elements, and schema connections represent the simple Association role construct and the Occurrences property of the simple Topic construct (as described in section 2.4.2).

2.4.4. Strict Topic Maps properties

- Subject locators, Subject identifiers and Item identifiers properties consist of collections of strings. CAM does not currently have a property interface implementation for editing a collection of strings, and one will need to be implemented.
- Value properties consist of a single string of text. CAM already has a text field property interface implementation, which will suffice for Value properties.
- Datatype properties also consist of a single string of text. While this string may be the identifier of any datatype defined in the XML Schema [14], three datatypes are more commonly used than the rest – String, IRI[3] and XML. CAM does not currently have a property interface implementation that facilitates easy selection of an item from a group of options, while also allowing additional options to be specified. One will need to be created.
- Type and Reifier properties require the ability to select a single Topic item in the Topic map. CAM does not currently have a property interface implementation allowing selection of a specific instance element from a workbook. One will need to be created.
- Scope properties require the ability to select many Topic items from the Topic map. CAM does not currently have a property interface implementation allowing selection of many instance elements from a workbook, so one will need to be created.
- As discussed in section 2.4.2, Topic name items and Variant items will be created and modified through instance properties of Topic instance elements. This structure is an unusual feature of the toolbox, and will require the development of a specialised property interface implementation.

3 The Internationalized Resource Identifier (IRI) is an extension of the URI to the full Universal Character Set

2.4.5. Simple Topic Maps properties

- Subject, Association and Role played properties all consist of a single string of text. CAM already has a text field property interface implementation, which will suffice for these properties.
- A simple Occurrence item's Type property is very similar to a strict Type property. The strict Type property's interface implementation, therefore, will also be used for the simple Type property.

2.4.6. Conversion operations and strict Topic Maps operations

The methods to convert between Topic map versions, and the operations mentioned in section 2.1.1, are all specific to the Topic Maps toolbox. Each will require an implementation of the menu item interface.

2.4.7. External interface requirements

CAM and its toolboxes, together, are designed to be a standalone platform for creating and analysing models. While it would be possible for the toolbox created in this project to use TinyTIM (a Java Topic Maps engine [15]), or to implement TMAPI (a Java Topic Maps application interface [16]), no real need has been identified for either. This toolbox, therefore, has no external interface requirements, other than those discussed above to interface with CAM.

2.4.8. Additional requirements

- Full documentation of code using Javadoc.
- Creation of a user guide for the toolbox.
- Good scalability of the toolbox – it should be possible to create large Topic maps without too much performance degradation.
- Generic property interface implementations to facilitate inclusion in future toolboxes.

3. Design and implementation

This chapter continues the discussion of CAM's structure (from section 2.2), and discusses the property interface implementations, two CAM palettes and menu item interface implementations developed for the Topic Maps toolbox.

3.1. CAM implementation

Consider CAM's structure again. Schema elements are instances of the `P3ElementSchema` class, instance elements are instances of the `P3Element` class, and schema properties are instances of the `P3PropertySchema` class. Instance properties are instances of a subclass of `P3Property` due to the vast differences between different properties. The meta-model and data-model object networks are accessible through public static methods of the `ASMNotifier` class, and so can be used by any class.

Instance connections are instances of the `PaletteElementConnection` class, a subclass of `P3Element`. This particular choice of class hierarchy is due to instance connections having instance properties in the same way instance elements do.

3.2. Property interface implementations

The property interface consists of three classes (Figure 3.1) – `P3Property`, `P3PropertyComponent` and `P3EntityInitialiser`. Subclasses of the `P3Property` class define the structure of the instance property's data, its initial value, and how it is saved and loaded. Subclasses of `P3PropertyComponent` define the user interface (UI) components (how the user interacts with instance property data) and how data is transferred between the UI components and the `P3Property` subclass. Subclasses of `P3EntityInitialiser` control non-trivial initialisation of instance properties, specifically any initialisation more complex than specifying an initial value.

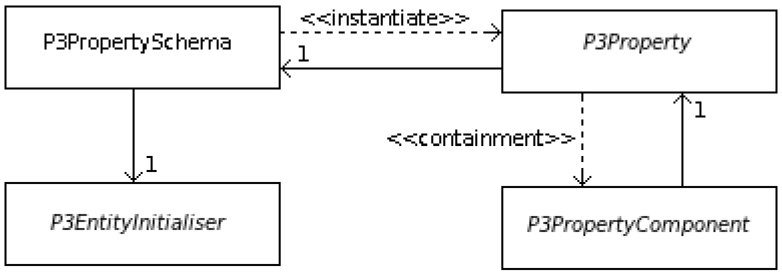


Figure 3.1: *The property interface.*

The `P3EntityInitialiser` class has an abstract `loadXML(Element)` method, which is called to interpret a property initialiser XML fragment declared in a palette file. This enables generic property interface implementations. For example, the drop-down box options for the `p3-list-selection-property` are specified in a schema element's palette file (e.g. in Listing 2.1), instead of being hard coded into the `P3Property` implementation. This avoids needing a new property implementation for every drop-down box property that defines a different set of options.

The `P3Property` class has a private `Object` data member, accessed through methods `getValue()` and `setValue(Object)`. Instance property data should be stored in this member to enable CAM functionality such as copying instance elements. The instance property data should be saved to an XML element when `P3Property.saveXML(Element)` is called, and loaded from an element when `P3Property.loadXML(Element)` is called.

The `P3PropertyComponent` class has a `JComponent` data member, accessed through `getComponent()` and `setComponent(JComponent)`, in which the user interface component is stored. The `JComponent` data member may be a `JPanel`, on which other UI components may exist. Stringified data is accessed from the `P3PropertyComponent` class by `getValue()` and `setValue(String)` methods.

All implementations of this interface must define a `P3Property` subclass. A `P3PropertyComponent` subclass is only required if the property defines a UI component, and a `P3EntityInitialiser` subclass is only required if instance properties need non-trivial initialisation.

3.2.1. List selection P3Element property

Recall from section 2.4.4, Type and Reifier properties require the ability to select a Topic instance element from the Topic map. When a Type or Reifier instance property is modified, there will always be a finite number of Topic instance elements in the Topic map. The UI component used to interact with Type and Reifier properties, therefore, is a `JComboBox` object, also known as a drop-down box or a list selection box.

This property interface implementation has been given the UID `p3-list-selection-p3element-property`. The following classes have been implemented for this property:

- `P3ListSelectionP3ElementPropertyInitialiser`, a `P3EntityInitialiser` subclass

- `P3ListSelectionP3ElementProperty`, a `P3Property` subclass
- `P3ListSelectionP3ElementPropertyComponent`, a `P3PropertyComponent` subclass
- A class hierarchy in the `p3ListSelectionP3ElementInitialiserElements` package (see Initialiser section)

Initialiser

To make the property as generic as possible, the drop-down box options are configured in a schema property's property initialiser XML fragment in a palette file[1]. These options consist of a sequence of static text options, and dynamic options formed from particular instance elements' labels.

The format of dynamic options' labels are specified in *label* XML element. Labels may contain literal text segments and segments equal to the value of a particular instance property. The *label* XML element is a mixed element – it may contain child elements embedded in text. Its child *propertyname* elements are replaced by the instance property value before displaying the label.

Considering the family tree example, if the *Person* schema element is given a *Partner* `p3-list-selection-p3element-property` schema property to record the person's current partner, *Partner* instance properties might display the option *Select partner:* followed by a list of *Person* instance element labels, formed from their *Title* instance property followed by their *Full name* instance property followed by the string *(Person)* (Figure 3.2 and Listing 3.1).

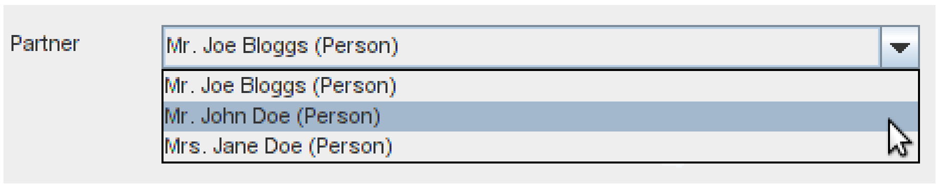


Figure 3.2: *A Partner instance property.*

1 An XML Schema has been created for this XML fragment and will be made available on the CAM website [10] when the toolbox is released

```
<initialiser>
   <option>Select partner:</option>
   <p3element element="Person">
         <label>
               <propertyname>Title</propertyname>
               <propertyname>Full name</propertyname>
               (Person)
         </label>
         <match type="include">
               <all/>
         </match>
   </p3element>
</initialiser>
```

Listing 3.1: *The* Partner *property initialiser XML fragment.*

If done in a single class, parsing and interpreting this complex XML fragment would be tricky to develop and maintain. A class hierarchy was, therefore, created in the `p3ListSelectionP3ElementInitialiserElements` package (Figure 3.3). Each concrete class is designed to parse, interpret and act upon a specific XML element.

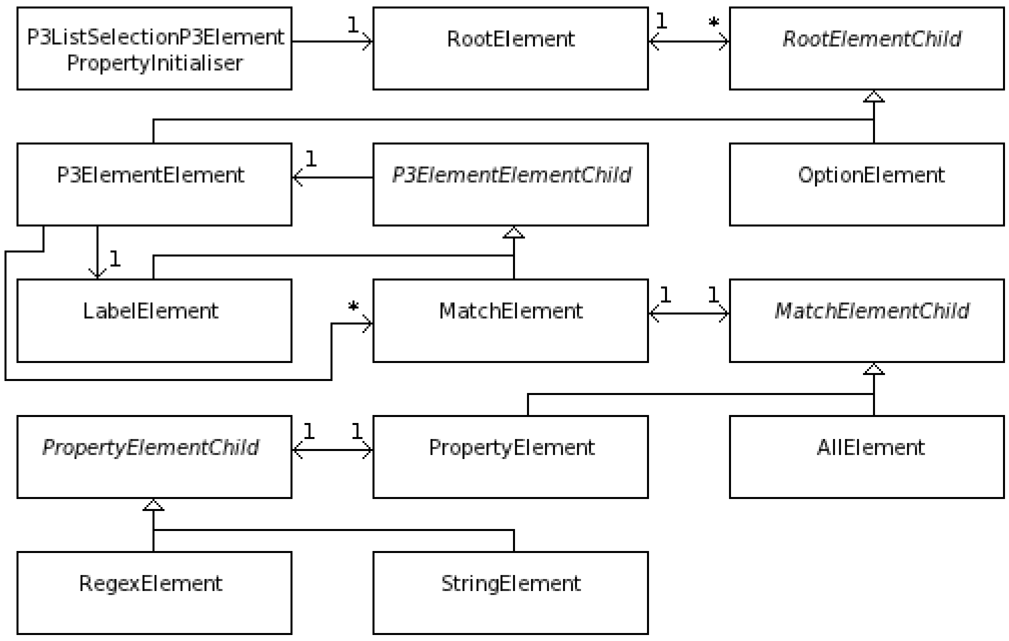


Figure 3.3: `P3ListSelectionP3ElementPropertyInitialiser` *class hierarchy.*

When CAM is executed, `P3ListSelectionP3ElementPropertyInitialiser` objects instantiate an object network (instances of classes defined in Figure 3.3) based upon the property initialiser XML fragment defined in the palette file. This object network is specific to a particular schema property, and is shared by all its instance properties across the CAM model.

Property

The alphabetically ordered options in the drop-down box can only be determined immediately prior to being displayed because instance elements may be created, modified or deleted at any time, changing the options in the list. Looping over the entire data-model to determine matches each time a list selection P3Element instance property is displayed is unfeasible – scalability to large data-models would be extremely poor. CAM, however, has a mechanism for notifying objects when an instance element is created, modified, or deleted. The static `ASMNotifier.dataModelHandlerObj` object is notified when the CAM data-model is modified, and passes the notification on to `DataModelChangeListener` listener objects.

The `MatchElement` class (Figure 3.3) implements the `DataModelChangeListener` interface. After being added to the `ASMNotifier.dataModelHandlerObj` object's listeners list, `MatchElement` objects are notified when instance elements are created, modified or deleted. On receiving a notification, they assess whether the action causes the instance element to change its matched status, from matched to unmatched or vice versa, and update the list of matched elements accordingly.

As noted in the Initialiser section, an object network containing a `MatchElement` object is shared by all instance properties of a particular schema property across all Topic map workbooks in the CAM model. `MatchElement` objects, therefore, maintain a separate list of matched instance elements for each workbook, the implementation of which is trivial. As a corollary of this, `MatchElement` objects require the identity of the particular workbook for which to construct the list of options. This can be trivially implemented in cases where the instance property is directly contained in an instance element. However, property interface implementations discussed in sections 3.2.3 and 3.2.4 may contain an instance property of this type[2], rendering the instance element indeterminable directly. The `ContainsListSelectionP3Element` interface was created, and is implemented by `P3Property` subclasses that contain a `P3ListSelectionP3ElementProperty` object to allow the instance element to be determined.

A selection in the drop-down box is of a particular instance element or static text option. Instance element labels may change when their instance properties are modified; the selection, therefore, cannot be tracked using the option text. Each instance element has a data item that uniquely identifies it across the model, accessible through the `getUID()` method. A

2 For example, Scope properties use a `p3-list-selection-p3element-property` instance property to select a Topic instance element to add to the Scope list.

string formed from this identifier is used to track selection, and is the property's data item. This means, however, the property's data item is not a user friendly string, a requirement for Scope property (discussed in section 3.2.3). The `P3ListSelectionP3ElementProperty` class implements the `P3StringValueIsKeyProperty` interface, providing a method to translate a property data value into a user friendly string.

Component

The Topic Maps standard places no restrictions on the uniqueness of Topic names and Variants across a Topic map, so many Topics may have the same display name. `JComboBox` objects do not normally allow duplicate entries. In the case of this implementation, however, it is reasonable to display duplicate entries – the user may create two Topic instance elements with the same name and both should appear in the list. To implement this, a wrapper class has been created for a `String` object, which performs a reference equality test when the `equals()` method is called.

3.2.2. Editable list selection property

Recall from section 2.4.4, Datatype properties require the ability to select an entry from a list, and to add an entry to that list. The visual component used to achieve this functionality is a `JComboBox` object in which the editable flag is set to true (by calling `setEditable(true)`). This allows the selection of options not in the list. An editable `JComboBox` object appears as a hybrid between a normal drop-down box and a text field (Figure 3.4).

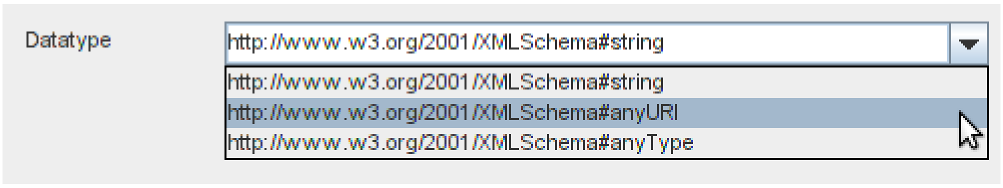

Datatype
http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2001/XMLSchema#anyURI
http://www.w3.org/2001/XMLSchema#anyType

Figure 3.4: *An editable `JComboBox` object (the Datatype property).*

This property interface implementation has been given the UID `p3-editable-list-selection-property`, and defines the following classes:

- `P3EditableListSelectionPropertyInitialiser`, a `P3EntityInitialiser` subclass
- `P3EditableListSelectionProperty`, a `P3Property` subclass

Initialiser

There are several possibilities for the scope of new options added to the list. New options might only be relevant to the instance property they are added to, or they may be relevant to all instance properties with the same schema property. To allow the palette creator freedom to choose the scope of new options, the property initialiser XML fragment contains a *scope* attribute. The two possible values of this attribute are *model* (indicating options should be identical for all instance properties of the particular schema property) or *property* (indicating additional options should not be shared across the model).

Property

Specifying a new option in the editable drop-down box causes the currently selected option to change to this new option, and causes the new option to be added to the instance property's list of options.

The schema property has a list of default options, specified in the property initialiser XML fragment. Each instance property with the scope *property* is given a new list of the default options on creation so that modifications to an instance property's list are not reflected in other instance properties' lists. Instance properties of a schema property with scope *model* share the same options list object, and additions to this list are seen in all instance properties with the same schema.

For the options lists to be re-created when a model is loaded in CAM, additions to each list must be saved in the data-model whenever a save action occurs. Each instance of a schema property with scope *property* saves its own additional options in the same location as its selected option. Schema properties with scope *model* coordinate their instance properties in such a way that options are saved to a single instance property. When the model is loaded from a save file, this instance property reconstructs the schema property's list of options.

Component

The only difference between the component of this property interface implementation and the component of the core CAM drop-down box property interface implementation is the editable flag. This vast similarity suggests the component class of this property interface implementation should be a subclass of the core CAM drop-down box property interface implementation component class, `P3ListSelectionPropertyComponent`. The visibility of the `P3ListSelectionPropertyComponent` class, however, is package-private, and the subclass cannot be created. Instead, component objects of this property interface are created by instantiating a `P3ListSelectionPropertyComponent` object, and setting the editable flag on the drop-down box contained in the object. Whilst not ideal, this solution avoids code duplication.

3.2.3. Single line set property

Recall from section 2.4.4, Subject locators, Subject identifiers, Item identifiers and Scope properties each consist of a collection of objects. These objects are Topic instance elements in the case of the Scope property, and `String` objects in the case of the other properties. While not identical, the similarities between the Scope property and the Subject locators, Subject identifiers and Item identifier properties can be exploited, and a single, generic property interface implementation created to satisfy the requirements of all these properties – the single line set (Figure 3.5).

The single line set property interface implementation may be used to edit any list of objects, providing each object has a meaningful string representation and providing another property interface implementation exists that can be used to edit items in the list. The Scope property's editor is a `p3-list-selection-p3element-property` property; Subject locators, Subject identifiers and Item identifiers properties use a `p3-single-line-string-property` editor. In accordance with the requirements of the Scope, Item identifiers, Subject identifiers and Subject locators properties, the single line set component does not permit duplicate entries in the list.

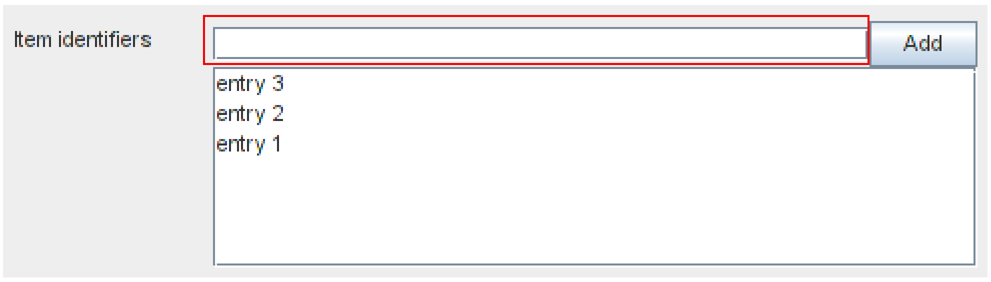


Figure 3.5: *Single line set instance property structure. The editor component is highlighted in a red box.*

The single line set property interface implementation has been given the UID `p3-single-line-set-property`, and comprises of the following classes:

- `P3SingleLineSetPropertyInitialiser`, a `P3EntityInitialiser` subclass
- `P3SingleLineSetProperty`, a `P3Property` subclass
- `P3SingleLineSetPropertyComponent`, a `P3PropertyComponent` subclass

Initialiser

While no limitations have been placed on the specific property interface implementations that may be used as a single line set editor, the user interface will only look reasonable in cases where the editor component has a similar height to a button (Figure 3.6).

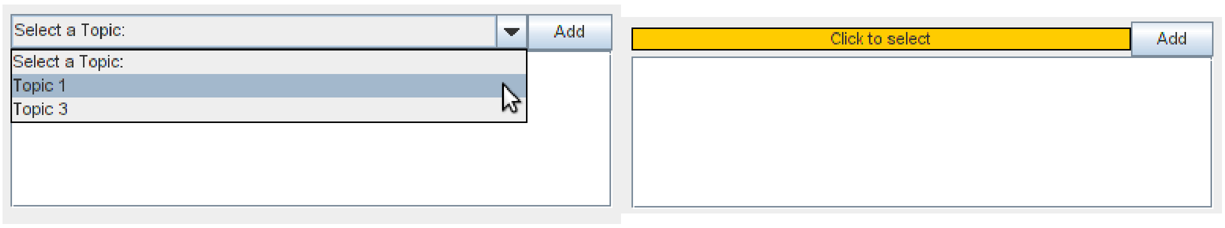


Figure 3.6: *Single line set instance properties with different editors. The left property has a* `p3-list-selection-p3element-property` *editor. The right proprty has a* `p3-color-property` *editor.*

The editor property interface implementation is specified by its UID in the property initialiser XML fragment, along with its own initial value and property initialiser XML (Listing 3.2). When a `P3SingleLineSetPropertyInitialiser` object loads an initialiser XML fragment, an editor schema property object is created. When single line set instance properties are created, they are given an editor instance property.

```
<initialiser>
   <editor module="Editor schema property UID goes here"
         defaultvalue="Editor schema property default value goes here">
      <initialiser>
         Editor initialiser XML fragment goes here
      </initialiser>
   </editor>
</initialiser>
```

Listing 3.2: *The single line set property initialiser XML fragment showing how the editor schema property is defined. The* defaultvalue *attribute and inner* initialiser *element are optional.*

Property

The data item of the `P3SingleLineSetProperty` class is a list of `String` objects, each of which is a previous return value of the editor `P3PropertyComponent` subclass's `getValue()` method. If the editor `P3Property` subclass implements the `P3StringValueIsKeyProperty` interface (like the `p3-list-selection-p3element-property` property), these data items are not user friendly strings. The `String` objects stored in the `P3SingleLineSetProperty` class's data item, therefore, may need to be translated into meaningful strings before being displayed in the `P3SingleLineSetPropertyComponent` component. If required, this translation is done using the `lookup(String)` method defined in the `P3StringValueIsKeyProperty` interface.

Component

The user interface component of this property is a `JPanel`, on which there is an editor component, an Add button and a `JList` object displaying the current list contents (Figure 3.5). A pop-up menu also exists and is displayed when the `JList` object is right-clicked.

When the Add button is clicked, the `P3SingleLineSetPropertyComponent` object retrieves stringified data from the editor component through the `P3PropertyComponent.getValue()` method. This data is translated into a user friendly string if necessary (as described above), and is added to the `JList` object if doing so does not cause a duplication.

An entry in the `JList` may be modified by highlighting an entry and selecting the edit option in the right-click menu. This causes the entry to be removed from the `JList` component. The value of the editor component this entry corresponded to is then determined, and the editor component set to this value by the `P3PropertyComponent.setValue(String)` method. The value is the actual string displayed in the list if the editor `P3Property` subclass does not implement the `P3StringValueIsKeyProperty` interface. If the `P3StringValueIsKeyProperty` interface is implemented, however, a reverse lookup is required.

3.2.4. Tree property

Topic name items and Variant items naturally form a hierarchy – each Variant item belongs to a Topic name item, and a Topic may have many Topic name items. This hierarchical structure suggests a `JTree` component, a component often used to display file hierarchies (Figure 3.7). `JTree` objects, however, display each node as a single string, and another method is required to edit Topic name and Variant item properties. Editing a node, therefore, opens a new properties dialog window through which, the instance properties of the Topic name or Variant item may be edited.

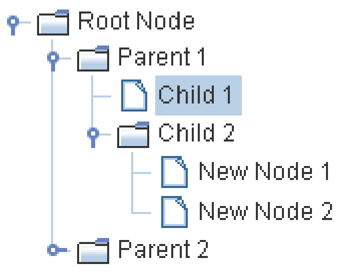


Figure 3.7: *A `JTree` component (from the Oracle Java tutorial website [17]).*

The tree property interface implementation is of a generic tree hierarchy, where tree nodes may have instance properties, edited through another properties dialog window. It has the UID `p3-tree-property` and comprises of the following classes:

- `P3TreePropertyInitialiser`, a `P3EntityInitialiser` subclass
- `P3TreeProperty`, a `P3Property` subclass
- `P3TreePropertyComponent`, a `P3PropertyComponent` subclass
- `P3TreePropertyNode`, instances of which represent tree nodes
- `P3PopoutPropertyDialogSchema`, a `P3ElementSchema` subclass. Instances of this class contain the schema properties of a tree node
- `P3PopoutPropertyDialogElement`, a `P3Element` subclass. Instances of this class contain the instance properties of a tree node

Initialiser

This interface implementation allows different schema properties to be defined for nodes at different levels in the tree. These schema properties are declared in groups in the property initialiser XML fragment (similar to Listing 3.2). Each group of schema properties is declared for a specific range of tree levels, along with the tree node label format for that range of levels (similar to the label discussed in section 3.2.1). The maximum tree height, and initial tree nodes are also specified in the property initialiser XML fragment.

When loading the property initialiser XML fragment, a `P3PopoutPropertyDialogSchema` object is created for each group of schema properties defined, and is given an object of each schema property in the group. These `P3PopoutPropertyDialogSchema` objects are stored in a map container, mapping tree levels to the `P3PopoutPropertyDialogSchema` object containing their schema properties. As multiple levels may map to a single `P3PopoutPropertyDialogSchema` object, a many-to-one map is used as the container. Java has no native many-to-one mapping class, so the idea presented in a Stack Overflow question [18] has been used to implement the generic `ManyToOneMappingSet` class based on a `TreeMap` object. This class implements the required addition and look up methods. Removal of mappings was not required or implemented.

Property

The data item used in this property is a `DefaultTreeModel` object, whose nodes are `P3TreePropertyNode` objects. Each node's user object is a `P3PopoutPropertyDialogElement` object instantiated from the `P3PopoutPropertyDialogSchema` object for the node's level. When instantiated, each tree instance property receives a deep copy of the initialiser's initial tree. A deep copy is an operation that duplicates an object entirely, including all of its data [19]. It is used in this situation to ensure modifying one instance property does not affect other instance properties.

The display string of a tree node is the return value of the node's user object's `toString()` method. The `P3PopoutPropertyDialogElement` class overrides the `toString()` method to return an entry's label (discussed in the Initialiser section).

Component

When an edit action is performed on a node, the node's `P3PopoutPropertyDialogElement` object's properties dialog window is opened to allow the user to edit the node's properties.

When the user finishes editing instance properties of an instance element, CAM may determine whether changes should be persisted or discarded depending on the action triggering the dialog to close. To avoid automatically persisting changes, the instance property's `DefaultTreeModel` object must be unaffected when a user modifies the `JTree` component. When a `P3TreePropertyComponent` object is created, therefore, it is passed a deep copy of the `P3TreeProperty` object's `DefaultTreeModel` object. If changes are to be persisted, the `DefaultTreeModel` in the instance property is replaced by the `JTree` component's `TreeModel` object.

3.2.5. Topic, Association and Occurrence display names

Various property interface implementations are required for Topic Maps instance elements to have appropriate display names. Topic display names are the Value property of one of the

Topic's Topic name or Variant items. Association display names are their Type Topic's display name. Occurrence display names are their Value property, followed by their Type Topic's display name in brackets. Association and Occurrence instance element display names should update when their Type Topic instance element display name changes.

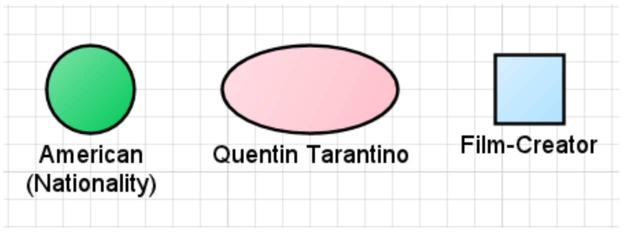


Figure 3.8: *The display names of Topic, Association and Occurrence instance elements. The pink oval represents a Topic, the blue square an Association, and the green circle an Occurrence.*

The additional property interface implementations required are:

- Hidden topic name property (UID: `tmaps-hidden-topic-name-property`) – The display name of Topic instance elements.
- Hidden occurrence name property (UID: `tmaps-hidden-occurrence-name-property`) – The display name of Occurrence instance elements.
- Tree name property (UID: `tmaps-tree-name-property`) – Used by Topic schema elements to modify Topic name and Variant items.
- List selection P3Element name property (UID: `tmaps-list-selection-p3element-controls-name-property`) – Used as the Type property of Association and Occurrence items.

Hidden topic and hidden occurrence properties are modified implicitly by the user, and do not show up on properties dialog windows. Their `getP3Component()` methods return `null`.

Topic display names

As mentioned above, a Topic instance element's display name property is a hidden topic name instance property. It consists, simply, of a `String` data item, which is modified by the tree name property implementation described below.

The generic tree property interface implementation is almost sufficient for a Topic name and Variant items property. It provides all required functionality, other than allowing a Topic name or Variant to be selected as the Topic instance element's display name. To leave a generic tree property implementation, a new property interface implementation has been created specifically for Topic name and Variant items – the tree name property. This implementation is based on the generic tree property implementation, with the addition of a *Set element name* menu item in the right click menu. Selecting this item causes the value of the Topic's hidden topic name instance property to change to the value of the selected Topic name or Variant node's Value instance property.

Each Topic's display node (the Topic name or Variant item that is the display name) must be tracked, so that modifications to the item's Value instance property are reflected in the Topic instance element's display name. The node cannot be stored as an object because changes are made to a copy of the tree properties data item as opposed to the data item itself (Component section of 3.2.4). The display name node is, therefore, tracked by its location, which is updated when addition and deletion operations are performed on the tree.

Association display names

The display name of an Association is also a hidden topic name property, which is set to the same object as the Association instance element's Type Topic's hidden topic name instance property. Association instance elements' display names, therefore, automatically reflect changes to the Type Topic's display name.

Association Type instance properties use the list selection P3Element name property interface implementation, which is based on the list selection P3Element property (described in section 3.2.1) with the following addition. Changing the selected option in the drop-down box causes the Association instance element's hidden topic name instance property object to change to the selected Topic instance element's hidden topic name instance property.

Occurrence display names

Occurrence schema elements also have a hidden topic name schema property, utilised by the Occurrence Type property in an identical manner to the Association's Type property. The Occurrence display name, however, is not set to the hidden topic name instance property. Instead, Occurrence display names are hidden occurrence name instance properties, which construct the display name string from the Occurrence's Value and hidden topic name instance properties.

3.3. Palette implementations

Putting together the simple Topic Maps and strict Topic Maps palettes simply involved creating palette files to define the schema elements and schema connections, and their schema properties (as discussed in section 2.4) and appearance.

Unlike many CAM model types, Topic Maps is very restrictive on the instance connections permitted. Both palettes' connection constraints only permit connections between Topic instance elements and Association instance elements, and Topic instance elements and Occurrence instance elements.

Topic Maps instance connections have no directionality. Connecting an Association to a Topic has the same meaning as connecting a Topic to an Association. The instance connections, therefore, provide the user with the usual CAM arrow head selection instance properties, but have the default value of no arrow head.

The Topic Maps standard does not define a symbolised representation of the constructs[3]. This toolbox employs colours and shapes to distinguish between different constructs.

3.4. Menu item interface implementations

CAM's menu item interface consists of a single class: `ASMApplicationMenuPlugin`. Subclasses of this class specify the implementation's operation, and the palettes for which it is a valid menu item. When CAM is loaded, a single instance of each menu item interface implementation is created. This instance is used whenever the menu item is executed.

3.4.1. Strict Topic map properties

Topic map items have two properties: Reifier and Item identifiers. In CAM, therefore, strict Topic Maps workbooks must have a Reifier property and an Item identifiers property. As CAM does not currently permit workbooks to have instance properties, Topic map properties are edited through a menu item defined in the `StrictTopicMapsPropertiesMenuPlugin` class. This menu item interface implementation makes use of the `TopicMapP3Element` class, a `P3Element` subclass used to store Reifier and Item identifiers instance properties.

When the menu item is selected, a `TopicMapP3Element` object's properties dialog window is

3 Graphical Topic Maps is a standard under development [2] for a Unified Modelling Language (UML) style representation of Topic Maps – consisting of text in boxes. This is not a symbolic representation, however, and is inappropriate for the CAM toolbox.

opened in a similar manner to an instance element's properties dialog window. The Topic map's instance properties may be edited in this window.

The `StrictTopicMapsPropertiesMenuPlugin` object used as the menu item does not change when the user switches workbooks. It, therefore, stores the `TopicMapP3Element` objects in a `TreeMap`, mapping each workbook UID to the `TopicMapP3Element` object containing its instance properties.

When a model containing a strict Topic map workbook is saved, the Topic map's Reifier and Item identifiers instance properties need to be saved. CAM, however, does not provide a method that permits menu item interface implementations to store data. A new property interface implementation has been created to achieve this functionality, which has the UID `tmaps-topic-map-p3element-container` and coordinates saving and loading each strict Topic map workbook's instance properties.

3.4.2. Conversion operations

Topic Maps version conversion operations have been implemented as menu interface implementations. When a conversion menu item is selected, a dialog window wizard opens to guide the user through the conversion process.

A conversion operation on a large Topic map may have a long execution time. To prevent the user interface becoming unresponsive, conversions are done in `SwingWorker` threads. When run, a conversion worker thread goes through the following process:

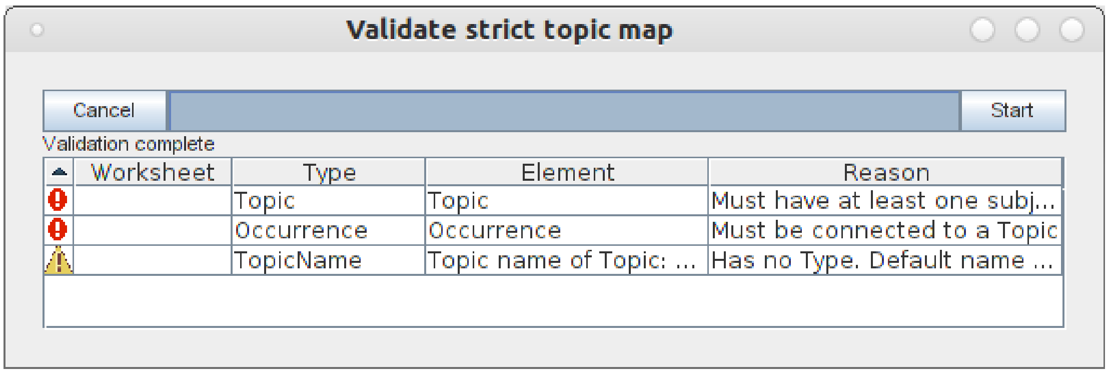
1. A new Topic Maps workbook is created of the opposite version to the workbook being converted.
2. Worksheets are looped over, and
    a) a new worksheet is created and added to the new workbook
    b) for each original instance element, a new instance element is created and added to the new worksheet
    c) for each original instance connection, a new instance connection is created and added to the new worksheet.
3. The new workbook is added to CAM's data-model.

Steps 2a-c encompass new instance property creation for worksheets, instance elements and instance connections respectively. Instance property creation is the only aspect of conversion operations dependant on the conversion direction. When creating a new instance property, an attempt is made to determine its value from the value of an instance property in the instance object being converted.

CAM uses visualisation data objects to control each worksheet's layout. All instance elements, instance connections and worksheets have visualisation data. For the converted worksheets to have the same layout as the original worksheets, the visualisation data of each new object must be a copy of the original object's visualisation data. Creating deep copies of visualisation data objects proved convoluted because there were many interconnections, all of which needed to be updated in the copy.

3.4.3. Validation operation

Validation is also an operation that has been implemented as a menu item interface implementation. When executed, a new frame is opened (Figure 3.9) for the user to interact with. This is a frame instead of a dialog to allow validation results to be viewed alongside CAM.



	Worksheet	Type	Element	Reason
●		Topic	Topic	Must have at least one subj...
●		Occurrence	Occurrence	Must be connected to a Topic
⚠		TopicName	Topic name of Topic: ...	Has no Type. Default name ...

Figure 3.9: *The validation operation user interface.*

If done in a single class, a Topic Maps validation algorithm would be tricky to develop and maintain. Instead, a more robust method has been implemented, involving the Topic Maps model discussed below.

Topic Maps model

The Topic Maps model is a class hierarchy, which has been created to represent each Topic Maps construct as a single class (Figure 3.10). Methods have been implemented to create a Topic Maps model representation of a Topic map from a strict Topic Maps workbook. When created in this manner, any violations encountered of the Topic Maps standard are recorded in the Topic Maps model for validation purposes.

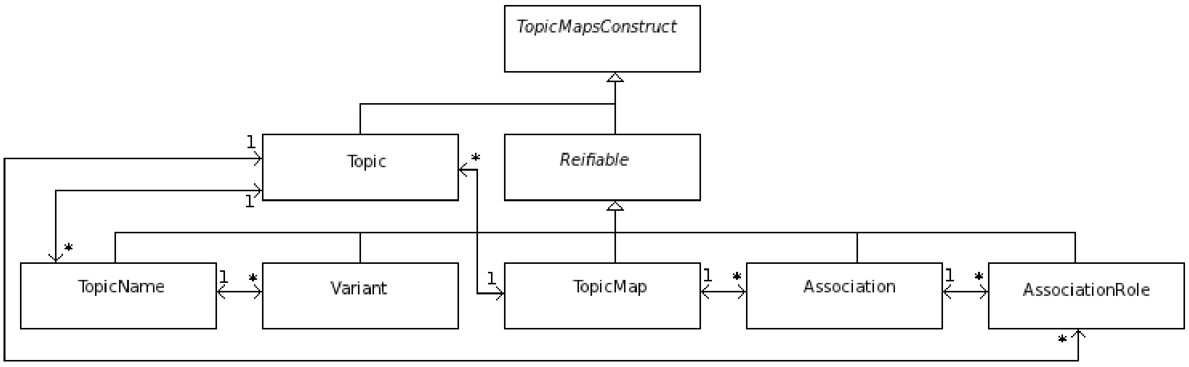
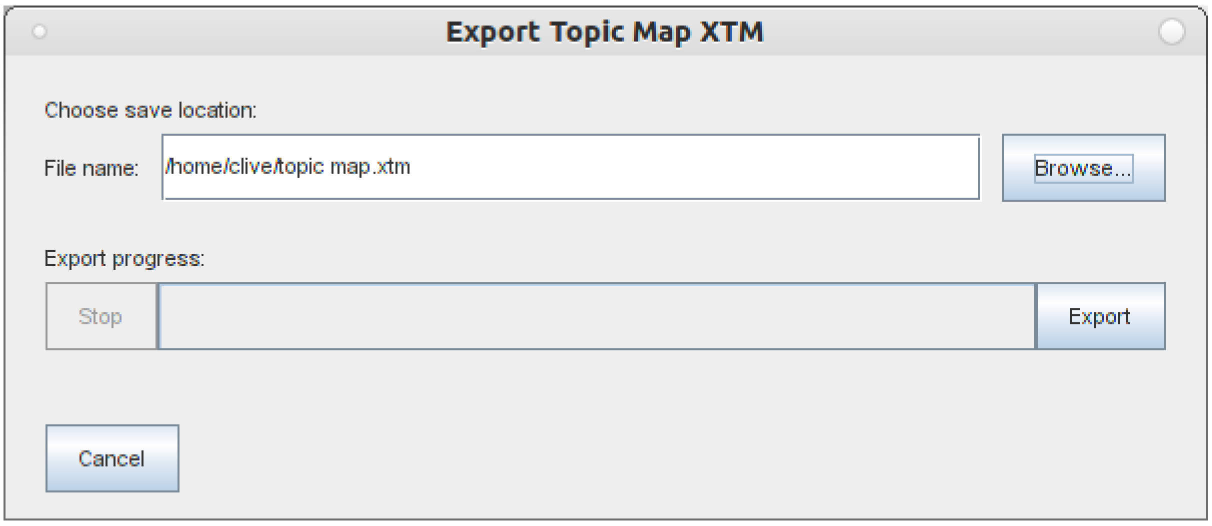


Figure 3.10: *The Topic Maps model class hierarchy.*

When run, the validation operation spawns a `SwingWorker` thread, which creates a Topic Maps model representation of the Topic map workbook being validated. After this model has been created, the validation table (Figure 3.9) is populated with all violations encountered during the Topic Maps model creation.

Double clicking an entry in this table focusses CAM on the instance element or instance connection in which the violation of the standard occurred, highlighting the offending element to the user.

3.4.4. Export XTM operation


Export Topic Map XTM

Choose save location:

File name: /home/clive/topic map.xtm

Export progress:

Figure 3.11: *The Export XTM dialog window.*

The Export XTM operation has also been implemented as a menu item interface implementation. When executed, a dialog window is opened (Figure 3.11), which allows the user to select the the XTM document's save location, and to trigger the save. These actions are performed on a dialog window instead of through a *Save as* dialog because the export operation may have a long execution time, during which the dialog window prevents the user from altering the Topic map.

The Topic Maps model created for the validation operation is also used in the export operation. Methods have been added to each Topic Maps construct class to create an XTM representation of the construct. The export operation executes as follows:

1. The Export XTM dialog window is opened.
2. After the Export button is clicked, a `SwingWorker` thread is spawned to create a Topic Maps model representation of the Topic map.
3. After the thread created in step 2 completes, another `SwingWorker` thread is spawned that generates an XTM document from the model created in step 2, and saves the XTM document.

Any errors encountered during this operation (for example, invalid save location errors) are reported to the user on the dialog window.

4. Evaluation

4.1. Achievement of objective

Recall from chapter 1, this project's objective was to design and implement Topic map representation and algorithms for use on the CAM software platform. This objective has been achieved successfully. Any Topic map that conforms to the Topic Maps standard may be represented in CAM as a strict Topic map, without loss of information. All critical Topic Maps operations have been implemented. The validation algorithm was considered the most important of these as it is the only method for the user to determine whether a strict Topic Maps workbook conforms to the Topic Maps standard.

The additional requirements on Javadoc and user guide creation (section 2.4.8) have also been satisfied. Both the Javadoc and user guide will be made available from the CAM website [10] when the toolbox is released.

4.2. Toolbox design

4.2.1. Performance

The CAM data-model increases in size as Topic maps grow. For scalability reasons, it is, therefore, important to avoid iterating over the data-model where possible. At no point during Topic map creation (a term encompassing instance element and instance connection creation, and modification of instance properties) is the data-model iterated over. This has been achieved using the `dataModelChangeListener` object (as described in section 3.2.1).

The conversion, validation and export operations will have longer execution times for large Topic maps, which, due to the nature of the operations, is unavoidable. During each operation, the user interface remains responsive (through appropriate use of worker threads), and ensures the user is kept up to date with the operation's progress (through appropriate use of progress bars).

The list of Topics in the Type, Reifier and Scope properties' drop-down box components will grow as Topic map size increases. As the user must be able to select any Topic in the Topic map, this growth of options is unavoidable. For ease of use, however, the Topics are ordered alphabetically in the drop-down box options.

4.2.2. Robustness to changes in CAM

While CAM is a stable application, development is ongoing. Therefore, the toolbox has been developed to be independent of internal CAM implementation details where possible. This aim has been achieved in all but two areas of the toolbox – the conversion algorithms (section 3.4.2) and the display name properties (section 3.2.5). The conversion algorithms are not robust to changes in CAM's visualisation objects' implementation details. The display name properties are not robust to changes in CAM's method of constructing a display name string from an instance property.

4.3. Testing

Unit testing was carried out as development progressed – each property interface implementation and menu item implementation was tested thoroughly after being developed. As CAM has no automated testing suite, all testing was carried out manually and over 100 Topic maps have been created to test various aspects of the different properties and menu items. Property interface implementations were tested by considering their weak points and creating Topic maps in such a way as to expose implementation flaws in these weak points. Topic Maps operations were tested on complex Topic map workbooks that used all CAM features, including instance element shortcuts, hyperlinked instance connections and layout groups.

After toolbox development finished, a brief usability test was carried out, and the following feedback was incorporated into the toolbox:

- The Value and Type sections of an Occurrence instance element's display name should be separated more clearly. This was taken into account – the Occurrence's Type is now displayed on a separate line to its Value.
- The default `JTree` folder and file node icons are not appropriate for Topic name and Variant item icons. The Topic maps specific tree property implementation (UID `tmaps-tree-name-property`, discussed in section 3.2.5) was, therefore, modified to display node icons as coloured squares instead of folder and file icons (Figure 4.1).

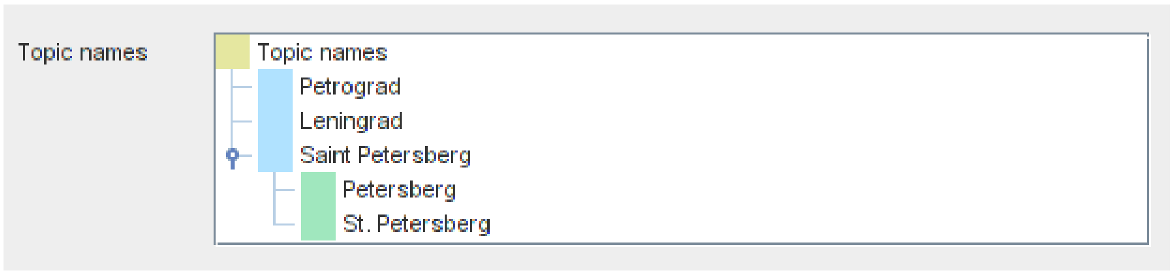


Figure 4.1: *The new Topic names property. Topic name items have blue icons, and Variant items have green icons.*

4.4. Challenges encountered

The main areas where challenges were encountered during development are outlined below:

- `MatchElement` objects using the `dataModelHandlerObj` object, and requiring the workbook identity when contained in another property (section 3.2.1).
- Tracking the Topic display name node (section 3.2.5).
- Saving data from a menu item interface implementation (section 3.4.1).
- Deep copying visualisation data objects (section 3.4.2).
- A new release of CAM part way through development causing the display name properties (section 3.2.5) to break.
- CAM coming with little code documentation, due to being a research funded project with few resources.

5. Conclusions and future work

A simple, visual Topic Maps application is a feasible concept. Topic, Association and Occurrence items may be represented by visual symbols, which may be connected using connectors representing Association role items and Topic-Occurrence relations. Topic name and Variant items may be represented as properties of Topic items.

As demonstrated in this project, CAM provides a suitable platform for a visual Topic Maps application. The Topic Maps constructs map well onto the CAM meta-model, and Topic Maps items and properties map well onto the CAM data-model. CAM also provides a suitable interface on which to develop Topic Maps operations.

I feel work has generally progressed smoothly on this project. With hindsight, however, I would probably have benefited from gaining a deeper understanding of CAM's implementation prior to starting development. This would have resulted in a more efficient development process, with less time spent tracing method calls through the core CAM classes.

I am currently working on an Import XTM operation, which will be completed before the toolbox is released. I am developing methods to instantiate a Topic Maps model (discussed in section 3.4.3) representation of an XTM document and to transform this representation into a CAM workbook.

There are a few remaining Topic Maps operations, which may be developed:

- Allow a strict Topic map to be merged (section 2.1.1), an operation that I will also complete before the toolbox is released. A merge operation involves the following steps:
    1. Create a Topic Maps model representation of the strict Topic map, to determine possible merges
    2. Display possible merges, and allow the user to select specific merges to perform
    3. Determine and execute the required operations on CAM's data-model to perform the selected merges
- Allow importing and exporting of Canonicalised XTM (CXTM) documents. CXTM is an XML based document defined as part of the Topic Maps standard [2] with the intended use of verifying the correctness of Topic Maps applications. Importing and

exporting a CXTM model would involve similar operations to importing and exporting an XTM document. After implementation, these operations should be used to verify the Topic Maps toolbox, possibly making use of the Topic Maps Test Suite [20].

Constraining and querying functionality (mentioned in section 2.1.1) could also be implemented.

The Topic Maps toolbox is scheduled for release on the CAM website [10] on 1st July 2012.

References

[1] **Witt, T.** (2009). *Diploma thesis: Conception and Implementation of a Visual Editor for Topic Maps.* Ludwig-Maximilians-Universität München.

[2] **ISO/IEC 13250** (2008). *Topic Maps*. International Organization for Standardization, Geneva, Switzerland.

[3] *Quentin Tarantino Wikipedia article* [accessed 28/25/2012]. http://en.wikipedia.org/wiki/Quentin_Tarantino

[4] *The filmreference.com Quentin Tarantino Biography (1963-)* [accessed 28/25/2012]. http://www.filmreference.com/film/96/Quentin-Tarantino.html

[5] *Saint Petersberg Wikipedia article* [accessed 28/25/2012]. http://en.wikipedia.org/wiki/Saint_Petersburg

[6] *Ontopia – the product* [accessed 28/05/2012]. http://www.ontopia.net/section.jsp?id=ontopia-the-product

[7] *Networked Planet Web3 Platform* [accessed 28/05/2012]. http://www.networkedplanet.com/Products/Web3/

[8] *Ontopia graphical visualization* [accessed 28/05/2012]. http://www.ontopia.net/page.jsp?id=vizigator

[9] **Wynn, D.C., Wyatt, D.F., Nair, S.M.T. and Clarkson, P.J.** (2010) *An Introduction to the Cambridge Advanced Modeller*, Proceedings of MMEP 2010. Cambridge, UK.

[10] *Cambridge Advanced Modeller* [accessed 28/05/2012]. http://www-edc.eng.cam.ac.uk/cam/

[11] *Cambridge Advanced Modeller toolbox documentation* [accessed 28/05/2012]. http://www-edc.eng.cam.ac.uk/cam/documentation/Toolboxes

[12] **IEC/ISO 19756** (2010). *Topic Maps Constraint Language.* International Organization for Standardization, Geneva, Switzerland.

[13] **IEC/ISO 18048** (2008). *Topic Maps Query Language.* International Organization for Standardization, Geneva, Switzerland.

[14] *XML Schema: Built-in Datatypes* [accessed 28/05/2012]. http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

[15] *Tiny Topic Maps engine* [accessed 28/05/2012]. http://tinytim.sourceforge.net/

[16] *Common Topic Map Application Programming Interface* [accessed 28/05/2012]. http://www.tmapi.org/

[17] *Java Tutorials: How to Use Trees* [accessed 28/05/2012]. http://docs.oracle.com/javase/tutorial/uiswing/components/tree.html

[18] *Stack Overflow: Using Java map for range searches [accessed 28/05/2012].* http://stackoverflow.com/a/1314708/406813

[19] *Object copy Wikipedia article* [accessed 28/05/2012]. http://en.wikipedia.org/wiki/Object_copy#Deep_copy

[20] *Topic Maps Test Suite* [accessed 28/05/2012]. http://cxtm-tests.sourceforge.net/

Appendix A: Risk assessment retrospective

A risk assessment was carried out prior to commencing this project, in which computer and monitor use were identified as the only potential hazards. This proved to be an accurate assessment. Retrospectively, the method of risk assessment carried out successfully identified all risks, and so would not be changed.