



## Configuration Management and Branching/Merging Models in iUML

Ref: CTN 101 v1.2

The information in this document is the property of and copyright Kennedy Carter Limited. It may not be distributed to any third party without the express consent of Kennedy Carter Ltd.

The terms "UML" and "iUML" refer respectively to the Unified Modelling Language and Kennedy Carter's MDA product suite and are used with the consent of the Object Management Group. All other trademarks are the property of their respective holders.

---

---

---

---

**K E N N E D Y C A R T E R**

[www.kc.com](http://www.kc.com)

## Table of Contents

1.	Introduction .....	3
2.	The Need for Configuration Management.....	4
3.	Components and Version Control in iUML .....	5
3.1	Components in xUML.....	5
3.2	Component Version Management.....	6
3.3	Component Exports.....	7
3.4	Offline Tools .....	10
3.5	Model Execution and Code Generation .....	10
3.6	Working with Components.....	10
4.	Interaction with External CM Systems .....	12
5.	Branching and Merging Models in iUML.....	14
5.1	Comparing Model Versions in iUML .....	14
5.2	Model Merge Capability .....	16
6.	Appendix A: Sample shell script invoking “offline” Modeller Utility .....	18
7.	Appendix B: iUML Model Merge .....	19
7.1	Invoking the Merge .....	19
7.2	How It Works.....	19
7.3	Configuring The Merge Process.....	20

## 1. Introduction

This document outlines the current support for configuration management and branching and merging of models in iUML as well as outlining expected future support.

Most of the features described here are present in the current production release of iUML (Release 2.3). However, some new features have been added in Release 2.4 which is scheduled to be available by the end of 2007. To avoid misunderstanding, these features have been marked with the following icon:



### Document History:

12 <sup>th</sup> August 2005	1.0	Initial Version
16 <sup>th</sup> February 2007	1.1	Information on iUML “diff” facility Further information on “offline” tools
12 <sup>th</sup> November 2007	1.2	Additional information about new iUML 2.4 features

## 2. The Need for Configuration Management

In the Executable UML approach, the models are, in one sense, “source code” for the system. As such, they must be treated as configuration items just like source code. It should be possible to baseline and freeze version and store these under formal configuration control. It must also be possible derive successor (child) versions.

Sometimes it will be required to derive two or more child versions to work on these independently. Such “branching” may be required for one of a number of reasons:

- While work on one branch (the “main line”) evolves the product towards its next major version, problems may be identified in the current production version in the field which must be addressed by “patches”. Such patches cannot be created in the context of the main line development since that will, at any given stage, include changes not yet ready for full release.
- Users may wish to continue semi-independent development tracks perhaps in different databases or geographical locations where it is expected that the developments will have minimal impact on each other.
- Even when working in the same location on developments that have a larger cross-impact on each other, developers may wish to work independently for some without being encumbered by multi-user restrictions and then subsequently resolve the impact of this.
- Core models may be sent out to multiple customers who each adapt them for their particular environment.

In order to support this kind of process, it is useful to have facilities to:

- Baseline and control multiple model versions and support the creation of branches
- Determine the differences between different model versions
- Perform automated or semi-automated merging of branched models into a single target model

In the following sections we discuss the current and future support for these ideas in iUML.

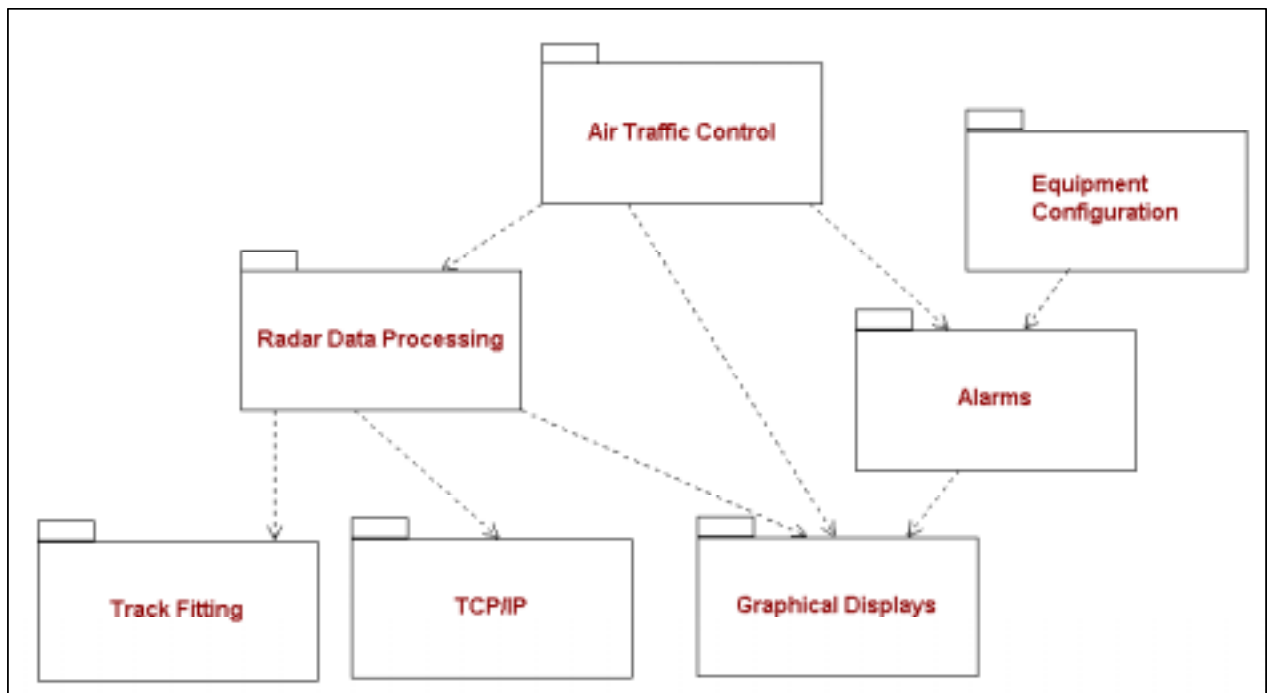
### 3. Components and Version Control in iUML

This section describes the facilities provided within the iUML modeller tool for managing the various components required for the xUML approach.

#### 3.1 Components in xUML

In the xUML approach, systems are described as an assembly of individual components called “domains”. Each domain deals with a different aspect of the system being described.

For example, an Air Traffic Control system might be described as an assembly of the following domains:



Each domain deals with a separate subject matter within the system and for each domain we will do **one** of the following:

- Create an executable UML model that describes the content and behaviour of the subject matter
- Create a model using another modelling language that describes the content and behaviour of the subject matter
- Select a pre-existing piece of software that implements the subject matter

Domains are thus the fundamental components from which we build the system. The nature of the partitioning means that work on each component can be carried out independently, although care must be taken to manage the interfaces between components.

To create a system, we define an assembly of domains. This assembly is termed a “Project”. The Project<sup>1</sup> defines the set of required domains as well as the connections (the “wiring”) between the domains.

Domains and Projects are thus components used to construct an xUML system.

Finally, it is possible to partition Domains into “Sub-domain Packages”. Each such package “owns” the definition of some of the Classes within the Domain. This enables different sets of classes to be modelled semi-independently of the others.

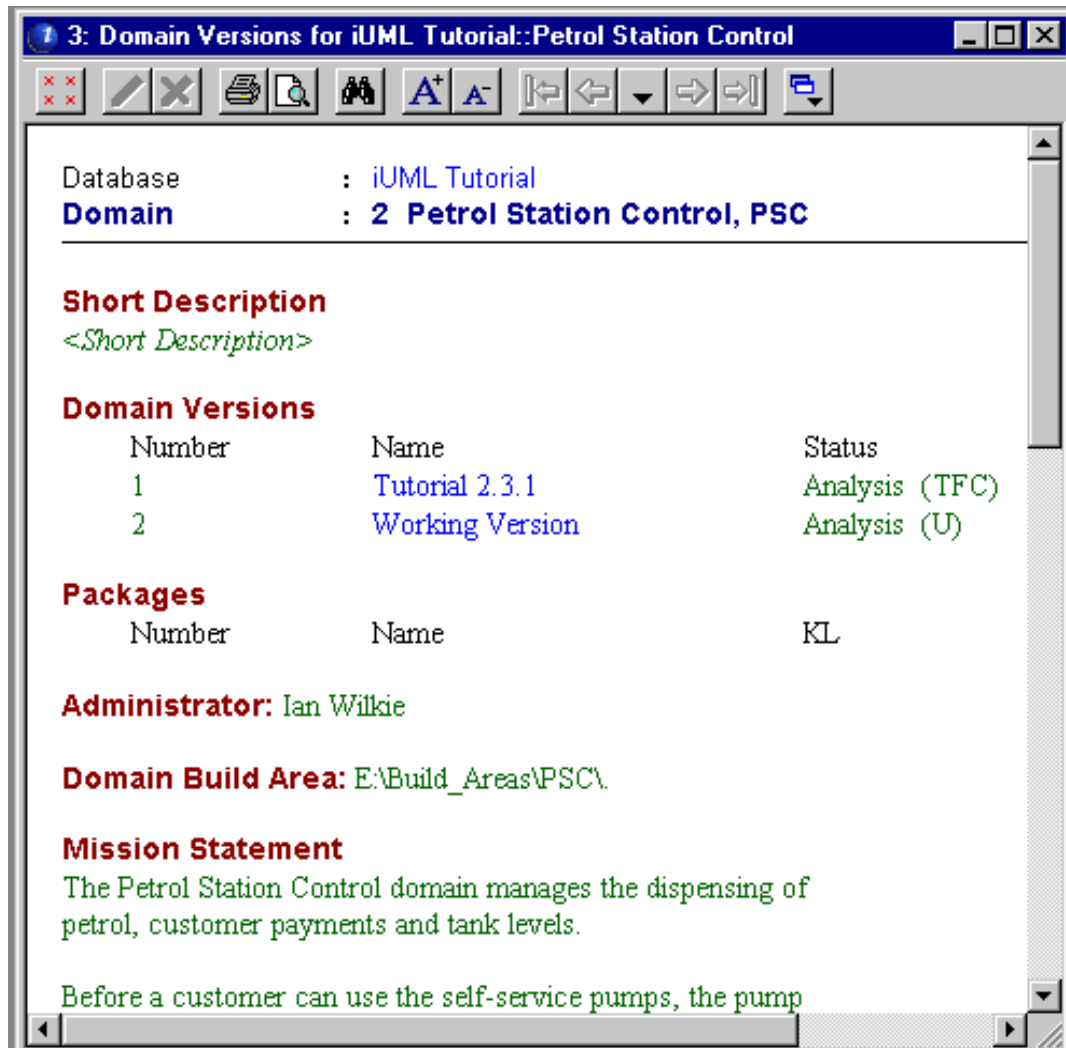
<sup>1</sup> Actually, a Project can contain the definition of a number of different assemblies. Each definition is termed a “Build set”.

### 3.2 Component Version Management

iUML has version management built-in to the Modeller tool. There are three types of versionable component:

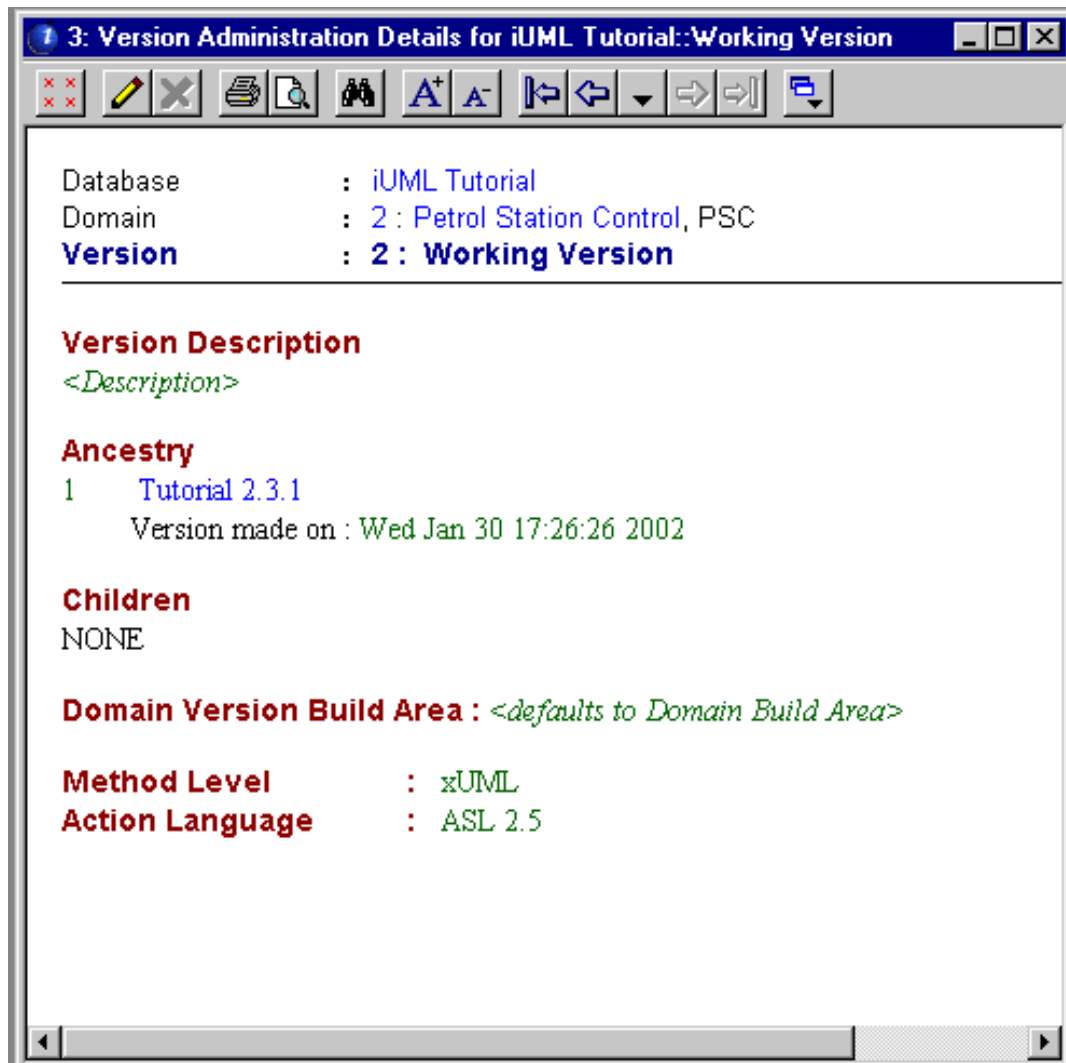
- Project
- Domain
- Sub-domain package

Each of these can be baselined and have child versions derived from them.



This screenshot from iUML shows the details of a domain model that has two versions. The first has been frozen and has a child (“TFC”) the second is unfrozen (“U”).

The following screenshot shows the details of the second version, including its ancestry and the fact that it has no children.



This versioning of components means that, for example, a version of a project (actually a build-set within a project) is defined as an assembly of versions of domains. Similarly, a version of a domain can be defined as an assembly of a set of package versions.

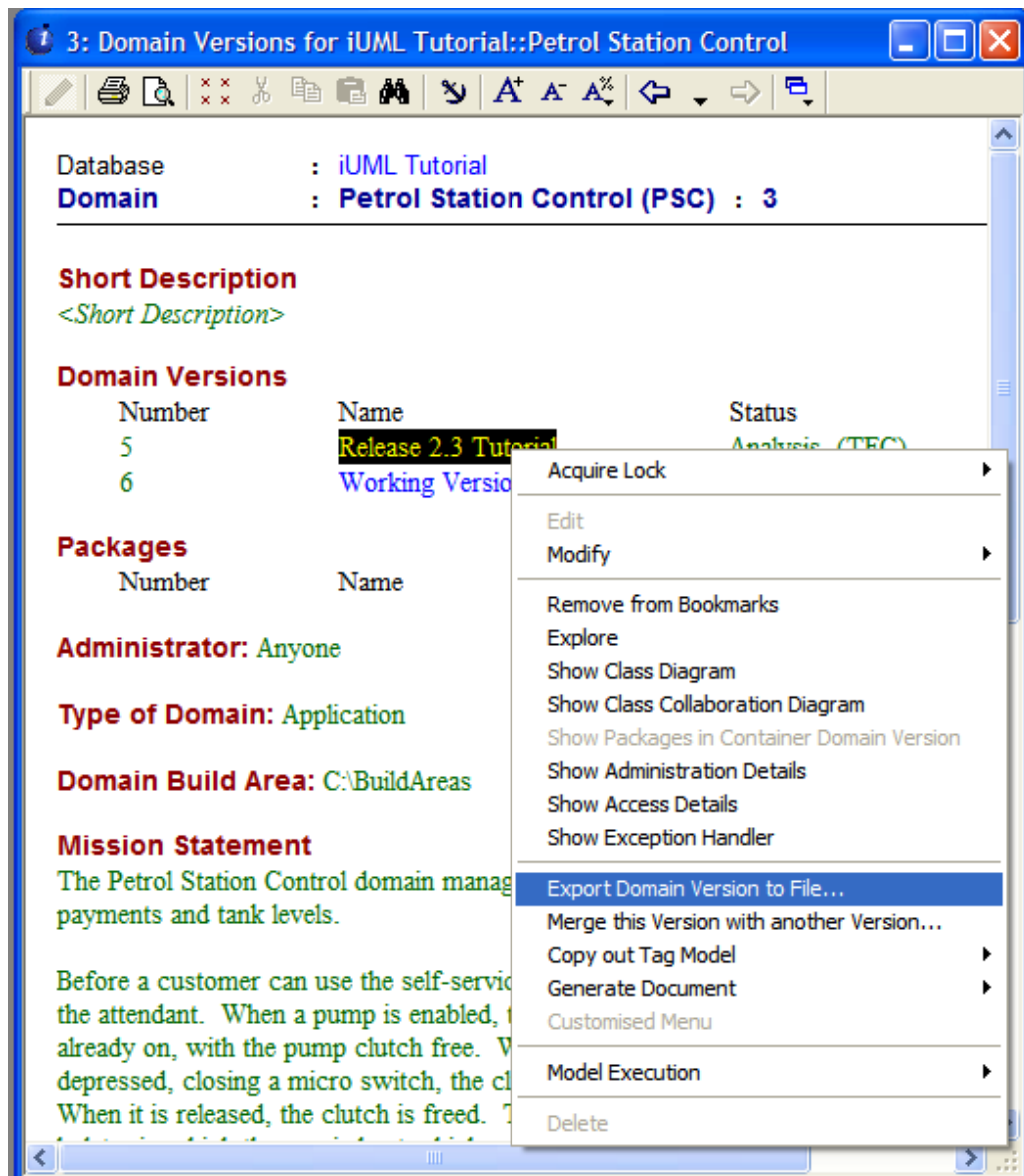
### 3.3 Component Exports

Versioned components can be moved from database to database through the import/export mechanism. When a component is exported, a copy of the information in the database is extracted into a defined file structure on the host file system. This structure (termed a “domain version export”<sup>2</sup>) can be moved between filesystems and then imported into another database.

This export/import can be achieved either through the User Interface of iUML (there are menu operations on versions of components) or by running “offline” tools.

<sup>2</sup> Or “Project Version Export”, or “Package Export”.

The screenshot below shows the export operation being invoked on a domain version.

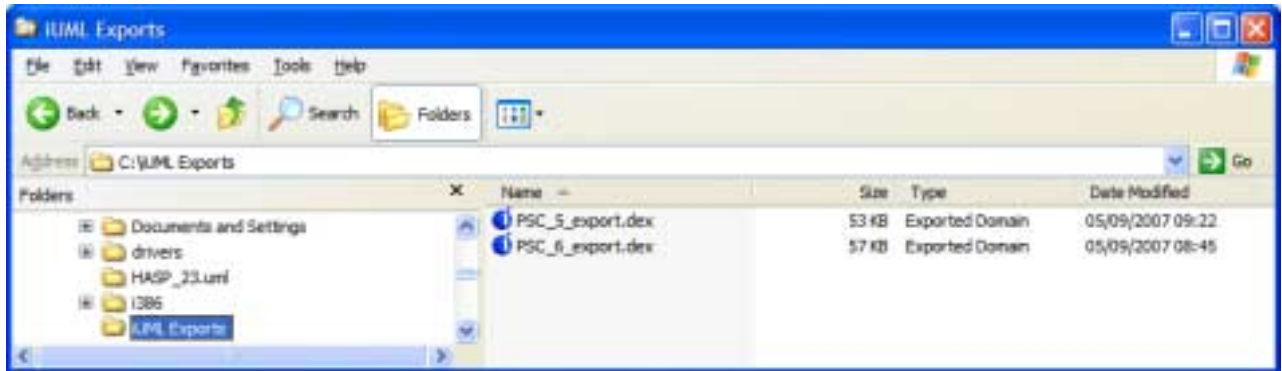


In all versions of iUML, the export can be produced in the following file structure:



however, as of iUML 2.4, exports can be produced as single files with a .dex or .pex extension<sup>3</sup>:

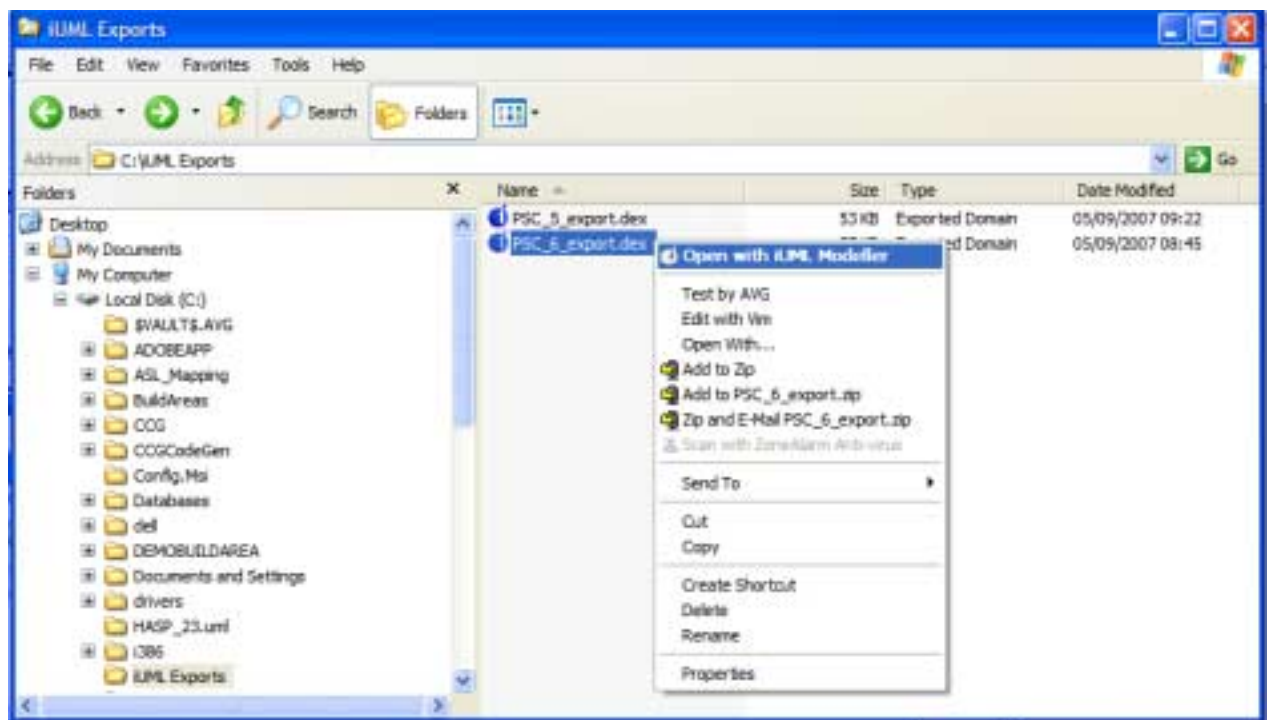
## iUML 2.4



These exports can be imported back into databases in a number of ways:

- Through menus in the interactive iUML tool
- Using one of the “offline” import tools described below
- Directly within a Windows browser by double-clicking on the export file:

## iUML 2.4



<sup>3</sup> “.dex” for domain/package exports and “.pex” for project exports.

### 3.4 Offline Tools

The component export/import operations can also be invoked using what are termed “offline” tools. These are utilities that can, with a single command line, be invoked to access an iUML Database and perform a manipulation on it.

These utilities can be invoked either directly from a command line in something like a Windows “command tool” or can be invoked from a user-written script. The offline tools can be called from scripts written in a variety of possible scripting languages like Perl, TCL, DOS bat (Windows) or Perl, TCL, Shell (UNIX). An example script is shown in Appendix A: Sample shell script invoking “offline” Modeller Utility”.

The following<sup>4</sup> offline tools are provided:

Tool	Function
export_project_version	Exports a specified project version from a specified database into a single export file
export_domain_version	Exports a specified domain version from a specified database into a single export file
export_package_version	Exports a specified sub-domain package version from a specified database into a single export file
create_database	Creates an empty database with a specified location and name
import_project_version	Imports the specified project version export into the specified database
import_domain_version	Imports the specified domain version export into the specified database
import_package_version	Imports the specified sub-domain package version export into the specified database

### 3.5 Model Execution and Code Generation

With the iUML tool suite it is possible to execute and test:

- Individual sub-domain package versions
- Individual domain versions (which may also be composed of one or more package versions)
- Individual build sets (composed of one or more domain versions)

This means that different components can be tested when in different databases at different geographical locations.

Note, however, that in order to execute a buildset, all of the components within it must be in the same database at the point at which the simulation is created.

With few exceptions<sup>5</sup>, target code generators provide the same capabilities. This, in turn, means that final target code generation must be performed from a single database containing all of the code for the build.

### 3.6 Working with Components

Given the scheme in xUML for assembling systems from sets of (possibly sub-divided) domains and the support provided for manipulating these components in iUML there are a number of different ways that development teams might choose to work.

<sup>4</sup> There are more offline tools than those that relate directly to CM and are described here. For example, there are tools which will generate reports. See the “iUML Modeller User Guide” section on “offline tools” for further details.

<sup>5</sup> For example, TA-6 does not support target code generation from a single domain version. However, the user can create a buildset that has only one domain version within it and generate code from that.

These range from performing the entire development activity within one iUML database at one location to the development of each component being carried out in a separate location.

### 3.6.1 Using a Single Central Database

In principle, there is no reason why all of the model data for a project should not be contained in a single database. A typical method of working is to use the built-in configuration management capabilities of iUML to baseline and up version components. Then, as production releases of software are made, the relevant component versions are exported and lodged under formal configuration control in the project's CM system. This ensures that a particular state of the models can be retrieved if required.

This approach has the advantage that:

- all of the data is in a single known location and housekeeping activities such as backups, database purging<sup>6</sup> and configuration management can be centrally managed.
- all of the components for the system are available for integration testing at any point

The disadvantages of this approach are:

- although each component can be modified independently within the database, configuration management actions (such as creating new models or versions of models) can be carried out only by one user at a time. On very large projects with may be inconvenient.
- All of the developers involved in the project must have high speed (LAN equivalent) connections to the file server holding the database and the connections must preserve full file locking semantics. In practice this usually means that all of the developers must be at the same geographical location.
- If a problem occurs rendering the database temporarily unavailable, all the developers on a project are affected.

This approach is therefore only suitable for comparatively small teams at a single location.

### 3.6.2 Using Multiple Databases

An alternative approach is to use multiple databases with one or more databases at each geographical location.

A typical mode of working is to have multiple teams, each of whom are responsible for a single domain or a set of domains. Each team will have its own development database and components will be exchanged between teams using component exports.

In this approach there will be one team who's job it is to manage the central integration of all of the domains in a single database for the purposes of integration testing and target code generation.

Typically, the individual development teams will submit tested and baselined version of their components to the central integration team according to the project schedule. At that point the exported component versions may be lodged in an external CM system.

The advantages of this approach are that:

- different teams can work independently, at different locations with no multi-user restrictions
- temporary unavailability of databases due to server or network problems affect only part of the entire development team

The disadvantages of this approach are:

- multiple copies of individual components exist at any one time

This latter issue can be addressed by careful process management and, perhaps, by using an external CM system to manage and control the exported components. It is even feasible to have the CM system has the primary repository and have users extract the exported components from the CM system each time they want to work on them. The ability<sup>7</sup> to open components directly makes this operation straight forward.

---

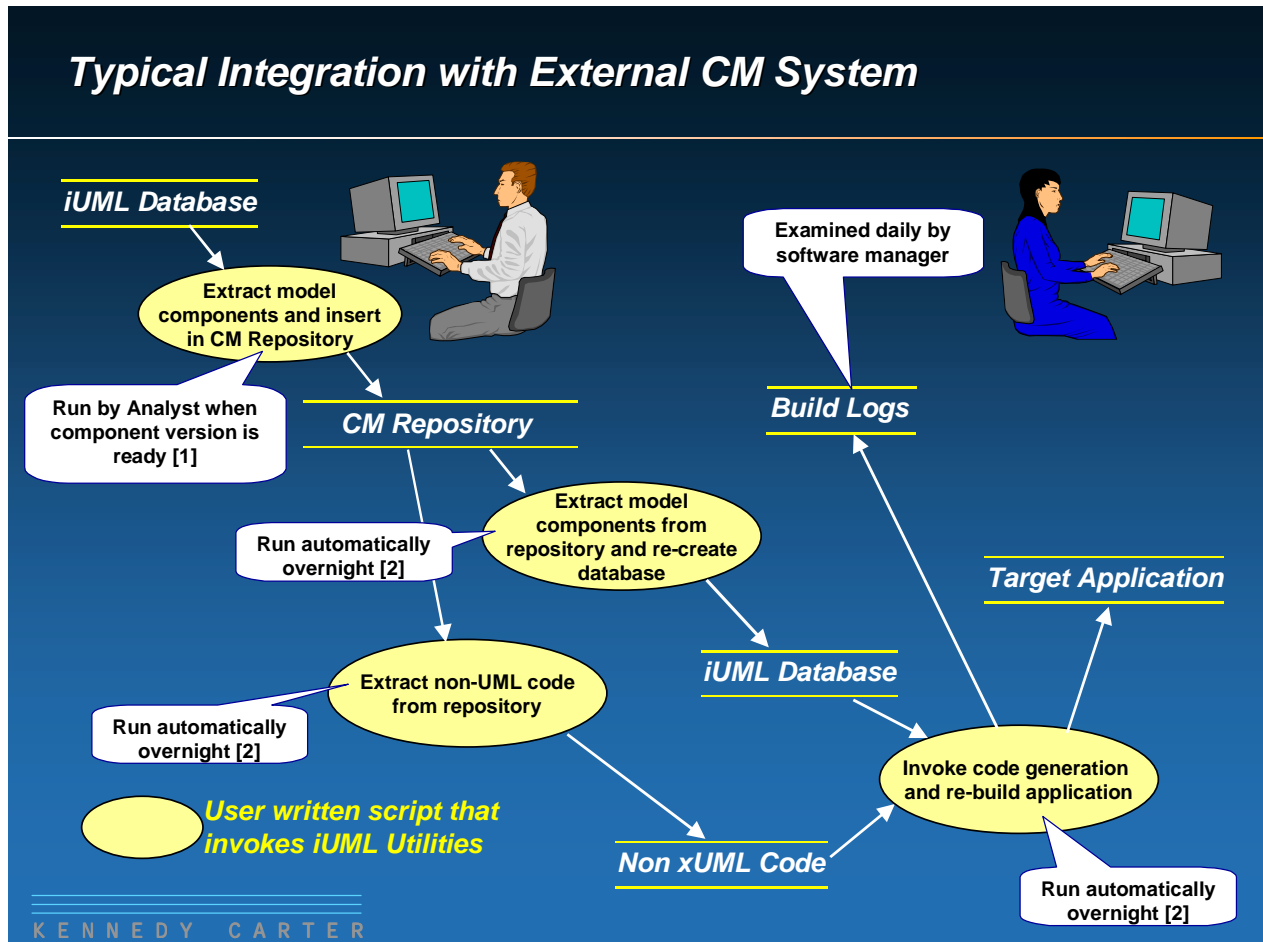
<sup>6</sup> Since every edit transaction on a database results in an addition to a trail of historical database states, databases should be "purged" periodically to avoid them growing too large. See the iUML Modeller User Guide for more information.

<sup>7</sup> This feature was added in iUML 2.4

#### 4. Interaction with External CM Systems

iUML does not have any off-the-shelf integration with any specific external Configuration Management system. Instead, it is expected that users will use the various tool features and offline utilities to create an integration that best suits the overall component development strategy they have chosen.

The following picture summarises one such integration<sup>8</sup> created by a real user of iUML:



In this organisation, a large system was partitioned into individual sub-systems each of which was the responsibility of a different development team. One such subsystem was developed using xUML and iUML. That subsystem was partitioned into domains in the standard way. The development of each domain was the responsibility of one or more developers.

Once the major system architecture had been created (i.e. the partitioning into sub-systems and then, for the xUML portion, into domains) each individual team implemented features according to their requirements. If interfaces with other components required change then the teams involved would coordinate with each other to achieve this.

At any given point, each team would be working on non-baselined versions of the models under their control. Once a set of changes had been completed, and suitable local tests had been passed, the models were baselined and submitted to the CM system (Activity [1] on the diagram).

At the same time, an automatic process would be run every night (Activities [2] on the diagram) that extracted all the baselined components from the CM system and re-built the entire system. Each morning the build logs were examined to check that the build had succeeded and, if the build had indeed been successful, the resulting software was submitted for testing.

This process was supported by a number of automated scripts (shown as the processes in yellow in the diagram) that:

<sup>8</sup> It should be emphasised that this is an **example** of a possible integration that suited the organisation in question. Each development organisation must first decide on the process that it wishes to adopt and then implement it.

- Use the CM related “offline” tools to export and store component versions within the CM repository or extract them later for reconstruction into a database

then:

- Use the reconstructed database to generate code and re-build the system.

Note that as well as it being possible to create scripts that access the offline utilities and code generators of iUML, such scripts can also be called through the iUML Modeller user interface through the mechanism of User Defined Menus. These are described in the iUML Modeller User Guide.

## 5. Branching and Merging Models in iUML

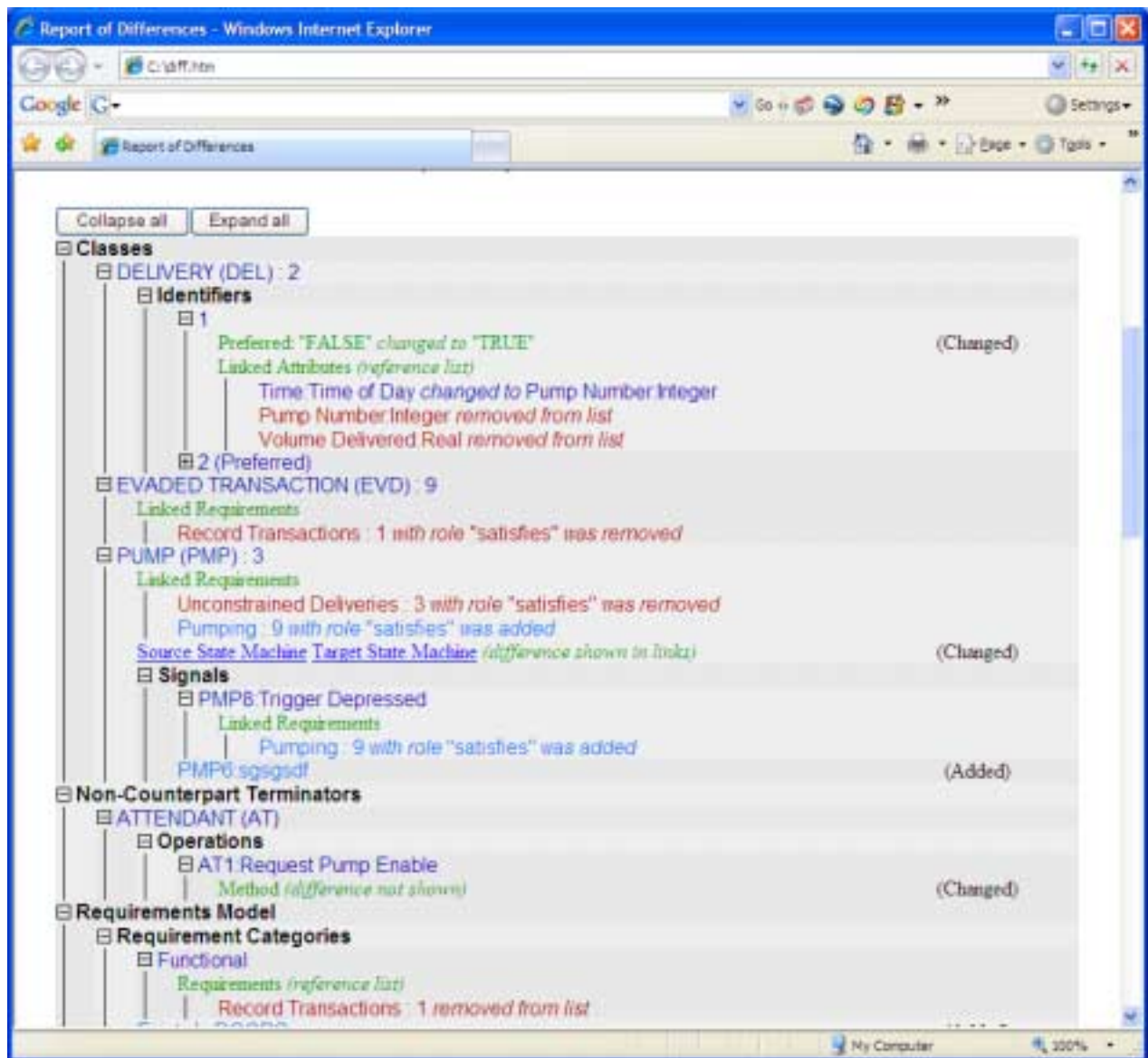
As described in Section 3.2, iUML permits users to branch component models. This may be done for a number of reasons:

- To support two or more users working on the same component without multi-user restrictions whether working in the same, or in different, databases.
- To support multiple threads of development such as “main line” vs. “patch” developments.
- To support the development of multiple different variants of a system

To support this activity, iUML provides facilities both to compare components and to merge them back together again. These facilities are described in the following sections.

### 5.1 Comparing Model Versions in iUML

iUML has the ability to compare different model versions (or even different models) in order to show the differences between them. When the comparison is invoked an HTML report is generated that displays the difference:



Note that this report is “active” in that controls are provided to be able to selectively hide and expose parts of the report. In addition, some differences are active links to other views. For example, the state machine comparison leads to a tabular presentation as follows:

Report of Differences in Table - Windows Internet Explorer

Target State Machine

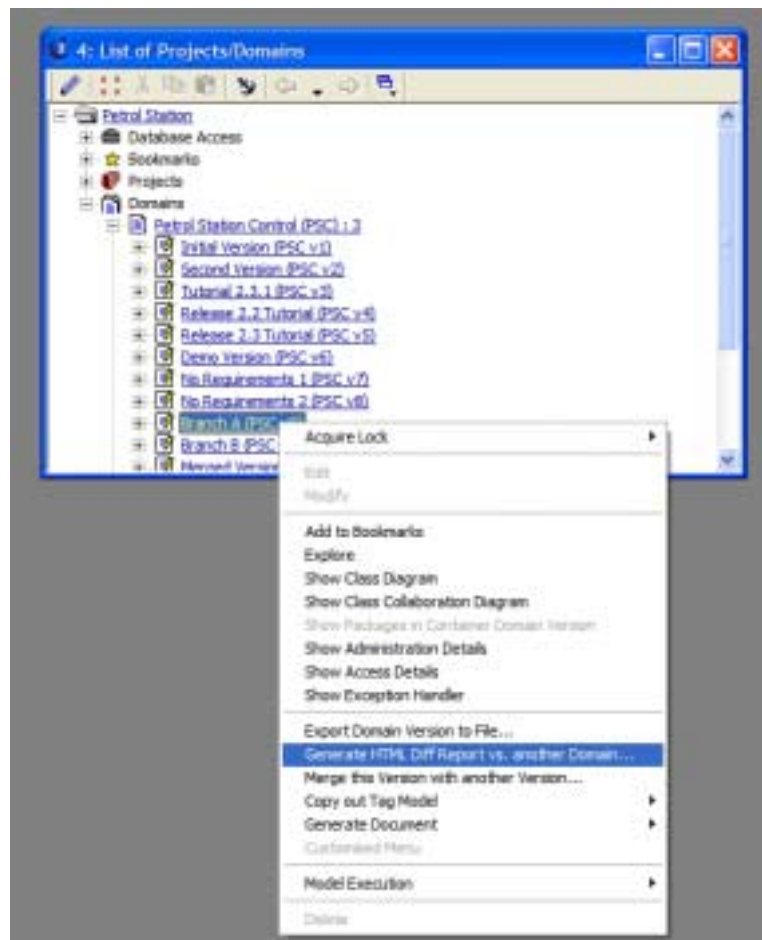
Table Key

- Cell Contents Have Not Changed
- Cell Contents Have Changed
- Cell Does Not Exist in Source Table
- Cell Does Not Exist in Target Table

	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released	PSFCustomer Released
Not Enabled : 0	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen	Cannot happen
Waiting For Customer : 1	Waiting For Customer : 2	green	green	green	green	green	green	green	green	green
Waiting Pump Enable : 2	green	green	Waiting For Customer : 1	green	Full Operation : 1	green	green	Ready To Pump : 2	green	green
Fuel Available : 0	green	green	green	green	green	green	Waiting For Customer : 1	green	green	green
Fuel Delivery Complete : 0	green	Waiting For Customer : 1	green	green	green	green	green	green	green	green
Ready To Pump : 1	green	green	Fuel Delivery Complete : 0	green	green	green	green	green	Pumping : 11	green
Pumping Enabled : 0	green	green	Fuel Delivery Complete : 0	green	green	green	green	green	Pumping : 11	green
Pumping : 11	green	green	green	green	green	green	green	green	green	Pumping Release : 0

Differences in textual attributes (such as Action Language Segments) are shown using the “WinMerge” tool<sup>9</sup> in the usual code oriented style.

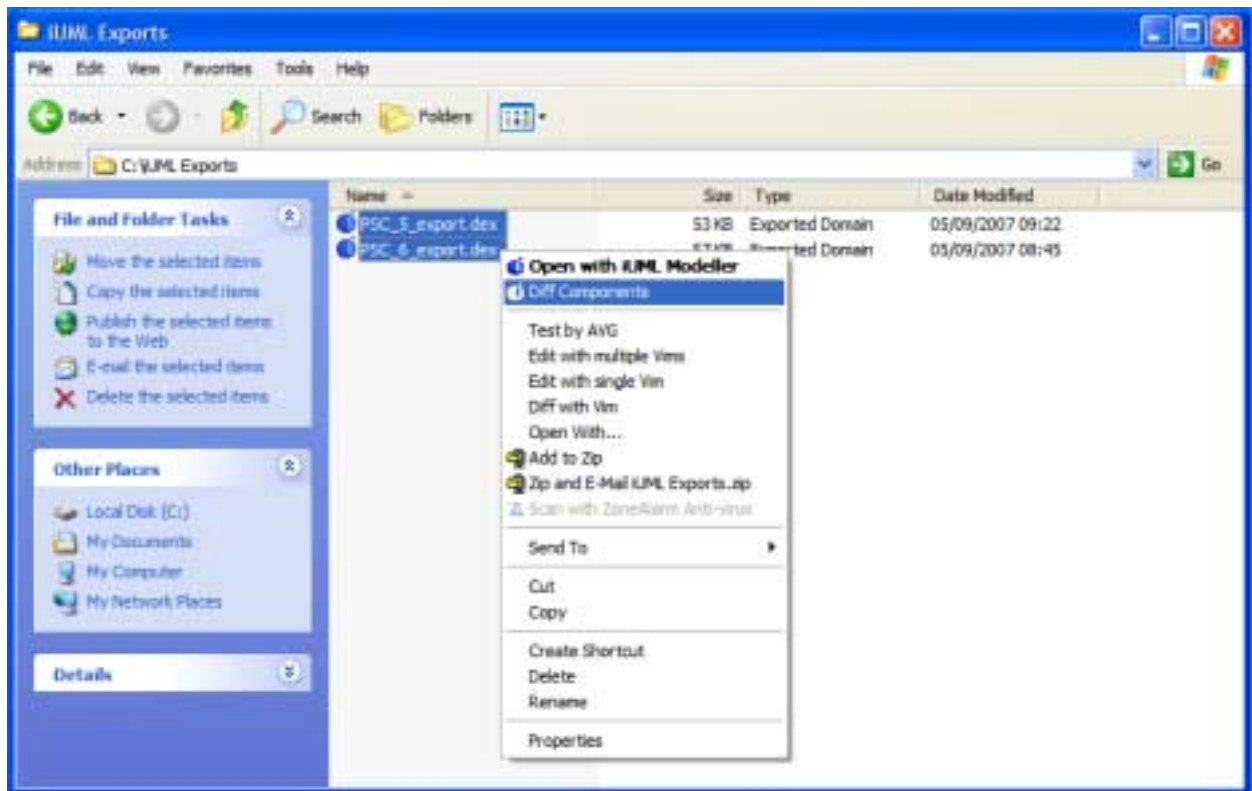
Difference reports may be obtained by comparing model components within a database by using the iUML Modeller user interface, for example:



<sup>9</sup> Or whatever textual comparison tool the user wishes to use.

In addition, model exports may be compared directly when viewed from a Windows browser:

## iUML 2.4

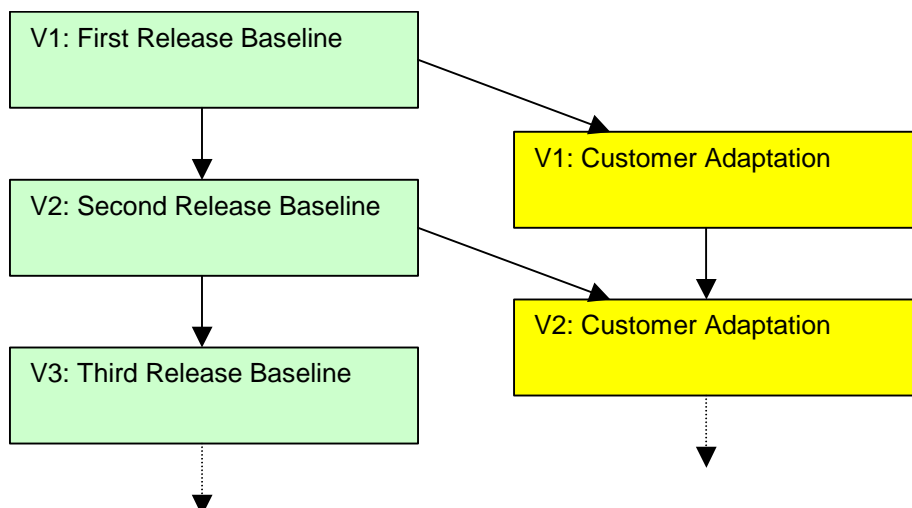


## 5.2 Model Merge Capability

iUML contains a powerful model merge capability that allows users to take two model versions and **merge** them into a single child version according to a set of rules attached to one or more of the models involved.

This facility is designed not only to support one-off merges but also to support repeated and automated merges of the same components.

Imagine the following pattern of model versions:



In this project, a core group are producing successive versions of a large product. As each version is baselined, it is sent out to various different “customer” organisations. Each of these customers must adapt the core models to their particular use of the application. Typically, this adaptation might take the form of:

- Replacement of lower level I/O related domains with suitable domains that address the customers actual hardware
- Adaptation of some bridges at a low level in the domain chart
- Changes to initialisation data in various domains

Clearly, it is desirable to avoid each customer organisation repeatedly performing a highly manual merge process. The current iUML Model Merge capability allows users to “Tag” one or more of the models involved so as to drive a “Merge” activity.

For example, users might tag some model elements in “V1: Customer Adaptation” to say, in effect, “do not overwrite” when merging in changes from “V2: Second Release Baseline”. Further, these tags **themselves** can be carried forward through successive customer adaptation versions. Thus, each successive version of the Customer Adaptation carries the rules that determine how changes are merged in.

With some planning, and appropriate model construction it will be possible to minimise the amount of manual work that has to be performed at each merge.

Of course, this approach can also be used for one-off merges (such as reincorporation of branches back into the main line), but the real power comes from being able to set up automated and repeated merges.

#### *Planned Future Enhancements*

Future versions of iUML will support the ability to control model merges interactively through a view that shows the differences between the models being merged. This will make the process of performing one-off merges easier.

## 6. Appendix A: Sample shell script invoking “offline” Modeller Utility

The following is a sample c-shell script that takes a domain version from an iUML database and inserts it into ClearCase with a suitable comment.

Notes:

- This is written in c-shell. It could be written in any appropriate scripting language such as Bourne Shell or even DOS batch script.
- The script assumes it is running on UNIX. On Windows, the script would have to be modified to use a scripting language supported on that platform.

```
#!/bin/csh
#
# iuml_baseline
#
# A script that takes a domain version from an iUML database and
# inserts it into clearcase.

if ( $1 == "" ) then
    echo "usage: iuml_baseline <database> <domain k1> <domain version>"
    exit
endif

# Assign arguments to named variables
set database=$1
set domain=$2
set version=$3

# Set up the iUML environment
#
# NOTE: If this script is run from an iUML
# user defined menu, the environment will already be available
source /home/tools/kc/iUML_2.00/uml_login.csh

# Run the iUML export utility and insert the result in a temp directory
export_domain_version $domain $version $database /tmp

# The export will have created a directory in /tmp called <domain>_<version>_export
set export_dir = $domain_$version_export
set export_file = $domain_$version_export.tar

# Pack up the directory into a single file
tar cvf /tmp/$export_file /tmp/$export_dir

# Now insert the file as a version of an element in a clearcase VOB
# (Script assumes we are running in a clearcase view already)
cd /home/development/iuimplvob
cleartool checkout -nc $export_file
cp /tmp/$export_file $export_file
set comment="Version inserted by iuml_baseline"
cleartool checkin -c $comment $export_file
```

If this script was called from a user defined menu in iUML then the parameters required:

```
<database> <domain k1> <domain version>
```

could be automatically passed in to the script. You can find out more about "offline" tools and user defined menus in the iUML Modeller Guide.

## 7. Appendix B: iUML Model Merge

### 7.1 Invoking the Merge

The merge is invoked from the *\*source\** (domain or project version) with the menu operation:

- Merge Project Version with...

or:

- Merge Domain Version with...

In each case, you will then be prompted to identify the **target** item (of the same type) into which model elements from the source will be merged.

You may merge items from multiple sources into the same target by repeatedly merging from the different sources.

### 7.2 How It Works

It is possible to use the Merge facility without any tuning at all, using only the default settings. However, the ability to configure the merge to suit your development strategy is a key feature. The effective use of this configurability requires a little background knowledge.

Merging is accomplished using the following algorithm (taking Build Sets as an example model element type):

- 1) The scope of the merge is defined through the selection of Source and Target Owner elements. In our Build Set example these are Project Versions (since Build Sets are 'owned' by Project Versions), selected by the user.
- 2a) Each element of the model element type in the Source Owner is looked at in turn, this element is the Source Element.
- 2b) The algorithm searches for element(s) identifiably the same as the Source Element in the scope of the Target Owner. It does this using identifying attribute(s) of the element type. In the case of a Build Set and most other elements, the name is used. Any elements found are Target Elements.
- 2c) If no Target Element is found, a new element is created under the Target Owner, and given the identifying attributes of the Source Element. This is then the Target Element.
- 2d) The data held directly in the Source Element (e.g. description field) is copied to the Target Element(s), overwriting any data held there.
- 2e) Links between the Source Element and other model elements not 'owned' by it (e.g. links to tags or requirements) are replicated on the Target Element(s). Links which do not exist on the Source Element are removed.
- 2f) This algorithm is called recursively on any substructure element types (e.g. a domain in a build set), using the Source Element and Target Element as the scoping elements in step 1 above.
- 3) Any elements of the Target Owner which were not referenced by the Merge, i.e. were not matched to Source Elements, are now deleted.

Note that the **exact** execution of this algorithm is controlled by Tags that can be attached to the source model or the target model or both. In particular, the Tag "Delete if Unreferenced" must be attached to a target element to cause step 3 to occur for a given item. If no such Tag is attached, an item that has not been matched to a source element will not be deleted. The "default" operation of the merge (i.e. with not merge control tags attached to either the source or the target) is thus a true **merge** of the two models with the definition of an element being taken from the source model in the case where the element existed both in the source and the target.

### 7.3 Configuring The Merge Process

Tuning of the Merge process is accomplished through the use of Configuration Tags, both attached to model elements and to define default behaviour. The full list of the Configuration Tags that are used is:

- Mergeable
- Not Mergeable
- Locked Against Merge
- Unlocked Against Merge
- Delete If Unreferenced
- Retain If Unreferenced
- Merge With
- DEFAULT Mergeable
- DEFAULT Not Mergeable
- DEFAULT Locked Against Merge
- DEFAULT Unlocked Against Merge
- DEFAULT Delete If Unreferenced
- DEFAULT Retain If Unreferenced

When the Merge operation is selected, iUML checks to see whether any Tag Groups containing these Tags exist in either your source or target project version. If so, then you will be presented with the choice to 'Merge With Default Settings' or 'Select Tag Group'. You will then be presented with a list of possible Tag Groups, prefixed with either "<", ">" or "<>".

Symbol	Meaning
<	Tag Group only present in source
>	Tag Group only present in target
<>	Tag Group present in source and target

The Tag Group you select will control the Merge. Only Tags from this Group will have any effect.

#### 7.3.1 Mergeable/Not Mergeable

Initial default setting: Mergeable

Active when attached to: Source Element

This Configuration Tag pair controls whether to attempt to merge the Source Element into the Target Owner.

This is checked at step 2a of the algorithm.

Tags Attached	Setting Used
Neither	Default/Inherited
Mergeable	Mergeable
Not Mergeable	Not Mergeable
Both	Default/Inherited (tags cancel each other)

If an element is Not Mergeable, then steps 2c-2e are not performed. Whatever setting is used becomes the new default for the call to merge substructure in step 2f.

### 7.3.2 Locked Against Merge/Unlocked Against Merge

Initial default setting: Unlocked Against Merge

Active when attached to: Target Element

This Configuration Tag pair controls whether the Target Element can be changed by the Merge process.

This is checked at step 2b of the algorithm.

<b>Tags Attached</b>	<b>Setting Used</b>
Neither	Default/Inherited
Locked Against Merge	Locked Against Merge
Unlocked Against Merge	Unlocked Against Merge
Both	Default/Inherited (tags cancel each other)

The tags' values are not used.

If an element is Locked Against Merge, then steps 2c-2e are not performed. Whatever setting is used becomes the new default for the call to merge substructure in step 2f.

Note that an element which is Locked Against Merge may still be deleted, if unreferenced, at step 3 of the algorithm.

### 7.3.3 Delete If Unreferenced/Retain If Unreferenced

Initial default setting: Retain If Unreferenced

Active when attached to: Target Element

This Configuration Tag pair controls whether to delete the Target Element if is unreferenced during a merge.

This is checked at step 2a of the algorithm.

<b>Tags Attached</b>	<b>Setting Used</b>
Neither	Default/Inherited
Delete If Unreferenced	Delete If Unreferenced
Retain If Unreferenced	Retain If Unreferenced
Both	Default/Inherited (tags cancel each other)

The tags' values are not used.

If an element has Retain If Unreferenced set, then at step 2e unreferenced links are not removed. Whatever setting is used becomes the new default for the call to merge substructure in step 2f.

Retain If Unreferenced also stops an unreferenced Target Element being removed at step 3.

### 7.3.4 Merge With

Initial default setting: none

Active when attached to: Source Element

The value of this Configuration Tag alters the value of the identifying attribute searched for in step 2b, in effect allowing the mapping of the Source Element onto a Target Element with a different name. This Tag may be attached multiple times to the same Source Element to map it to multiple Target Elements.

Step 2c is not performed when this Tag is used.

### **7.3.5 DEFAULT Configuration Tags**

These Tags are used to reconfigure the default settings that every Merge starts with. They work by simply being present in the Tag Group which is used to tune the Merge. Contradictory defaults cancel each out in favour of the initial default values given above.