Title: A Model-Centric Approach for the Management of Model Evolution in Chemical Process Modelling

Authors: Chiou Peng Lam, Huaizhong Li, Dong Xu

PII: S0098-1354(07)00027-0
DOI: doi:10.1016/j.compchemeng.2007.01.010
Reference: CACE 3404

To appear in: *Computers and Chemical Engineering*

Received date: 9-1-2006
Revised date: 25-1-2007
Accepted date: 30-1-2007

Please cite this article as: Lam, C. P., Li, H., & Xu, D., A Model-Centric Approach for the Management of Model Evolution in Chemical Process Modelling, *Computers & Chemical Engineering* (2007), doi:10.1016/j.compchemeng.2007.01.010

A Model-Centric Approach for the Management of Model Evolution in Chemical Process Modelling

Chiou Peng LAM, Huaizhong LI, Dong XU [1]

School of Computer and Information Science
Edith Cowan University
2 Bradford Street, Mount Lawley, WA 6050
Australia

[1] This author was on leave from Shanghai University, Shanghai, China. This author was a visiting researcher at Edith Cowan University when the paper was written.

Abstract. This paper explores the development of an automatic model centric version control approach for managing the evolution of chemical models and to support model reuse. Unlike traditional versioning which is text-based, the basis of versioning in the proposed approach is based on structural changes of the chemical models. An implemented prototype tool incorporating the proposed approach and the use of an example XML-based representation of chemical models was used to illustrate the associated concepts. Some results associated with a case study are presented. Given that chemical modelling tools like HYSYS has just delivered an XML infrastructure that aims to support collaborative engineering, this proposed technique is timely and relevant. The benefit of the approach is that it provides a way of automatic storage and retrieval of the chemical models as well as the management of the different versions of the model as it evolves, thus allowing chemical process engineers to concentrate on the modelling and simulation.

Keywords: Model evolution management, version control, process modelling, XML, Model Reuse

1. Introduction

Process modelling supports activities such as process design, process control and the optimisation of process design/operation of complex unit operations. Although there are advanced modelling tools available commercially, the most time-consuming step in a process modelling project is the enormous amount of effort required in model formulation and configuration. This is because:

- the modelling process is often poorly understood and is highly dependent upon individuals (i.e. background and work/tool preference). As indicated in Foss's study (1998), a sufficiently precise model specification may not be available at the start of project. The modeller at this point, has to decipher information from various sources

and often do not have the complete set of data for physical property and reaction rate calculation.

- the rigorous modelling of chemical processes/systems is time consuming and error prone owing to the enormous number/variety of chemical process units and physico-chemical phenomena.

Foss *et al* (Foss *et al*, 1998) have stated that there are a lot of similar requirements between designing complex software systems and designing chemical process models. Some of these requirements are:

- Explicit specification of model fidelity and functional characteristics of a model-based application at the beginning of the project.
- Documentation reflecting the rationale behind the design.
- Management of processes that are used to produce artefacts in the system.
- Management of processes that control changes to artefacts in the system.
- The reuse of existing design artefacts.

Software configuration management, a technique in software engineering, is a discipline for organising and controlling evolving complex software systems. Configuration management practices originated in the 1950s. They were first used in the development of military hardware systems and the US. Army, Navy and Air Force have all developed a number of configuration management standards, such as DOD-STD 2167 (Department of Defense, 1995). Since then, configuration management systems aim to provide automated support for configuration management tasks and are capable of storing all artefacts related to a software system. In addition to the versioning facility, they usually also support three additional layers of functionality; a *configuration control layer*, a *process management layer* and a *problem*

reporting layer (Caballero, 1994). The *configuration control layer* maintains information about the artefacts related to the software system, providing information about each version and their interrelationships. The *process management layer* stores the lifecycle (the states of the artefact such as check-in, check-out or release states) of each type of artefact found in the system. Lastly, the *problem reporting layer* supports bug and enhancement tracking, providing a mechanism for linking bug reports and enhancements to the respective modifications. Commercial or free software configuration management tools and systems (e.g. Arch, BitKeeper, Clearcase, CVS, Subversion) that automate some or all of these activities are currently available (Better SCM Initiative, 2006).

In the development lifecycle for chemical process design, model knowledge is frequently lost due to improper documentation or lack of mechanisms for systematic management for the evolution of chemical models. Frequently, in carrying out the numerous chemical process simulations, directories within a file system have been adopted to serve the purpose of distinguishing different versions in the evolution of the process models as the designer attempts to obtain the optimised solution. Existing models are either hard to be found, or hard to be reused because of insufficient documentation associated with the different versions of the models. In addition, there are situations where activities associated with a simulation may need to be carried out across different modelling and simulation tools and owing to proprietary formats, the models have to be reconstructed. As a result, repetitive modelling is a common practice in chemical process research and industry (Schopfer, von Wedel, and Marquardt, 2000).

This paper proposed an automatic model-centric approach to the management of the evolution of chemical models in an attempt to address some of the issues raised by Foss *et al* (1998) as

well as to provide a technique that may be use to support chemical model reuse. The development of a tool that integrates proven techniques from the area of configuration management for chemical process modelling is significant, as it will aid the management of the evolution of the chemical models. Given that chemical modelling tools like HYSYS has just delivered an XML infrastructure that aims to support collaborative engineering, this proposed technique is timely and relevant. In addition, it will improve the quality of the model and reduce the cost as well as the time required for process modelling. This is possible as such a tool will enable:

- The development and storage of the various versions of a chemical model to be supported automatically. The purpose, assumptions and decisions related to model versions can be documented in a manner that will emphasise the interrelationships of these aspects.
- Reusability and modifications of existing chemical process models will be easier, as the rationale behind the original design which if documented by the user will be available for subsequent use. The substantial cost involved in developing and validating any non-trivial process model for similar processes can be reduced.

The remainder of this paper is organised as follows: The remainder of this section discusses significant related work and Section 2 describes various concepts associated with the proposed approach. The proposed technique was implemented in a prototype tool. Section 3 describes and demonstrates its use in managing the evolution of chemical models using a case study. This is followed by a discussion in Section 4 and the conclusion in Section 5.

1.1 Background

Almost all aspects of plant design and operations, ranging from basic process design to plant operation, control and optimisation involve mathematical process models directly or indirectly. In the past, there is a strong coupling between applications and process models that are described using some specific mechanisms (Pantelides *et al,* 1994), for example

traditional steady-state *flowsheeting* packages are mainly process simulators where the model of each unit operation has a form specifically designed to facilitate this function. While there has been substantial development in process modelling (Ponton, 1995), existing limitations still include:

- An inability to reuse parts of models, out of and sometimes even within their original context (Ponton, 1995, Jarke *et al*, 1996).
- A lack of mechanisms for documenting information and decisions during the design phase (Bogusch *et al*, 1996; Foss *et al*, 1998; Ponton, 1995, Subrahmanian *et al*, 1993).
- A lack of mechanisms for controlling and documenting the history of the various versions of a model built during a modelling project (Jarke, 1996; Westerberg *et al*, 1997).
- A lack of support for team efforts in model development (Bogusch *et al*,1996).

Recent progress in addressing these limitations has been limited. The development of chemical process models from scratch is still costly, time-consuming and error-prone. The ability to reuse models or parts of models would greatly reduce the resources required for model development. In order to have the reuse capability it is necessary to develop mechanisms for documenting information and decisions during the design phase and mechanisms for controlling and documenting the history of the various versions as a model evolves. A number of systems attempting to incorporate these characteristics are discussed in the following sections. An important point about a number of these systems (with the exception of the IMPROVE project (Marquardt *et al*, 2004) is that the simulation environment is implemented from scratch by incorporating some form of methodology to facilitate the

capturing of modelling decisions, dealing with issues related to storage and retrieval of data as well as truth maintenance.

MODKIT (Bogusch *et al*, 1996) is a modelling environment that supports systematic development of chemical process models. This tool differs from others such as SPEEDUP or ASCEND in that it supports the documentation of:

- informal knowledge such as model assumptions and limitations, specification of degrees of freedom and model initialisation, and
- the purpose of each version of a model and the interrelationships between the various versions.

MODKIT incorporates a hypertext system that creates comprehensive documentation linked to the modelling objects, thus recording all assumptions, decisions, and experiences gained in the model development process as well as background knowledge relating to the physico-chemical phenomena. This system implements a method known as Issue-Based Information Systems (IBIS) in order to support the documentation facility. In this system the documents are broken into small pieces known as hypertext nodes and these nodes are link by hypertext links. The nodes can contain information while the links are used to create the semantic structure of a document (See Bogusch *et al* (1996) for details).

n-dim (Westerberg *et al*, 1997) is another general modelling environment that allows documentation of design knowledge in a meaningful way so that it can be used by other users, thus aiding the reuse of existing models. It also supports the integration of other design tools providing one environment for various design activities. Other process-based design-support systems currently under development include IDIS and épée/KBDS. IDIS is designed to

capture the exploration of design alternatives, design rationale and design constraints (Goodwin *et al*, 1994). Although this system is able to maintain the chronological order of design, it does not provide chronological versioning of design discussions or constraints. KBDS is a system that captures design history by associating design constraints, alternatives investigated and models in a common environment. An assumption-based truth maintenance system (ATMS) is used to enforce design constraints and it uses épée as an integration layer in order to provide access to external tools.

In terms of some existing techniques that support reuse of existing process design artefacts, Surma and Braunschweig (1996a, 1996b) described a case-based retrieval approach that allowed the retrieval of the best matching flowsheet from an existing library of Process Flow Diagrams. Two similarity measures, the *aggregation similarity* that measures the flowsheets as a set of components and the *connection similarity* that quantifies the connections between components in the flowsheet were used. The basis of retrieval was determined by calculating the similarities between two flowsheets that were represented as graphs. Flowsheets are retrieved based on the criteria of either: possessing the largest similar sub-graph to the query flowsheet or possessing the smallest least-cost set of transformation operations required to transform it to the query flowsheet. A review of similarity measures which can be used for case-based design in process engineering have been detailed by Avramenko and Kraslawski (2006). They discussed various approaches where the degree of similarities between features is calculated based on structural comparison, quantitative distance, location in a hierarchy as well as qualitative comparisons of defined terms.

Since CAPE-OPEN there is various emerging research (Marquardt *et al*, 2004) aimed at improving process modelling tools to address various existing limitations. One of the projects

(Jarke *et al*., 1999) associated with CAPE-OPEN represents an initial effort towards providing support for modelling teams, although its overall objectives also deal with the other limitations listed above. The IMPROVE research project (Marquardt *et al*, 2004) is aimed at developing tools and solutions for supporting collaborative design processes and to promote interoperability amongst tools in chemical engineering. Amongst the many outcomes from this project is a software tool, AHEAD (Heller, 2004), which support the management of collaborative design of chemical processes; ROME which is a chemical model repository (Wedel, 2000) for storing symbolic models in a neutral or in a proprietary format associated with its commercial simulator and Cheops which supports the runtime integration and linking of models from different modelling environments into a single flowsheet. However, to the knowledge of the authors, no concerted effort has yet been made to develop an approach or a tool that addresses the issue of automatic management of the evolution of chemical models using structural changes of the models as the basis for versioning while supporting model reuse and documentation of the model-development decision-making process.

2. Model-Centric Evolution Management in Chemical Process Modelling

2.1 Motivations for Model Evolution Management

It is well-known that heterogeneous modelling and simulation tools for process modelling exclusively use proprietary formats to represent process models. These modelling and simulation tools are incompatible in general; therefore, the models developed using these tools are not inter-operable. Unfortunately, it is a common practice in process design to employ different tools for different purposes. Additional efforts often have to be devoted to convert a model developed in a tool into a format which is usable in another tool (Schopfer *et*

al., 2000). Therefore, it is desirable that a tool-independent modelling representation is available to facilitate information exchange between these different tools.

Model reuse refers to use of available design artefacts, such as model knowledge and tool specific implementations, in the development and maintenance of a process. Obviously, model reuse across different modelling and simulation tools can save time and resource for the process design. However, chemical model reuse in a process design lifecycle is difficult. Owing to the proprietary formats used to represent models in the model development tools, current model reuse practice is largely confined to within the same model development tool only. Besides issues relating to the incompatibilities of the modelling and simulation tools which hinder the reusability of the models, currently there are also no well-established standardised procedures for developing chemical process models. Hence individual engineers and organizations may develop and maintain models in their own cryptic ways. A neutral exchange language for process modelling may help standardizing the model development procedure.

Model integration heavily involves model reuse in different ways. Model can be reused either *a-priori* where the reusable models are designed and implemented with specific consideration for integration, or *a-posteriori* where the models are reused and integrated almost arbitrarily (Marquardt *et al.* , 2000). It is clear that there is a key issue related to model reuse and model integration:

Model Evolution Management: models have to be managed in a systematic way for promoting reuse and evolution in the process development lifecycle.

To promote the model interoperability and to enable model management, it is desirable that the models within a tool are represented using a neutral exchange language. One of the promising possibilities for a neutral exchange language is the Extensible Mark-up Language (XML). As stated by von Wedel (von Wedel, 2002; von Wedel, 2003), XML has the great advantage to represent process models which can be easily represented using the XML flexible meta model. Many existing tools can be used and integrated in the process development environments to process XML data. XML representation of the models enables systematic model management. Models represented in XML format can be managed and validated using a corresponding XML schema for process modelling.

Applying XML to process modelling has attracted many research attentions in chemical engineering research society recently. For example, a XML based data model, CapeML was developed for CAPE-OPEN which aims to facilitate standards for Process Modelling (von Wedel, 2002). Based on CapeML, XML format was exclusively used to manage heterogeneous models in chemical process engineering (von Wedel, 2003). XML format was employed to construct common data models that conform to the plant data models in conceptual process design (Seuranen *et al.*, 2005). An exchange language in the form of a XML schema was proposed in Li and Lam (2004) for process modelling and model management to promote interoperability of process models and to enable systematic management of process models. Another XML-based model representation named Modelith was developed by Linköping University, Sweden for model transformation and exchange in several domains, including chemical engineering (Larsson et al., 2002, Modelith, 2006). In addition, a XML-based software architecture for configuration and usage of chemical process simulation models was introduced by Karhela (Karhela, 2002) to facilitate model reuse and to improve configuration co-use between different simulators.

Furthermore, the Collaboratory for Multi-scale Chemical Science (CMCS) team, which consists of research leaders from U.S. government laboratories and academic institutions, is developing new approaches that facilitate collaborative sharing of chemical model data and analysis codes. One of the recently focused interests for the CMCS team is to develop XML based formats for large detailed chemical models to enable model data exchange (Schuchardt *et a.*, 2005).

Applying XML to process modelling and simulation has also attracted great attentions in chemical engineering industries. For example, the new HYSYS 3 modeling and simulation environment from Aspen Technology now delivers an XML infrastructure in the hope that the infrastructure will support collaborative engineering and promote exchange of engineering knowledge. HYSYS 3 uses XML technology for case linking, case management, model reuse and case browsing. The new XML capabilities in HYSYS 3 enable chemical processing companies to improve engineering productivity and enhance decision-making (HYSIS, 2003). As stated in the HYSYS User Guide (HYSYS, 2003):

"The most significant development within the HYSYS 3 series in this regard is the delivery of XML (eXtensible Mark up Language) technology. The range of possibilities that this opens up are significant, but some of the immediate benefits are:

- *The ability to store all (or part) of the user's inputs and specifications in XML to allow re-building of the case.*
- *The ability to store parts of an existing simulation case in XML and have it read into another case, either augmenting or overwriting the definitions within that case.*
- *The ability to store simulation case results in an XML format to allow post processing of simulator data, taking advantage of the wide range of XML technology being developed within the software industry.*

- *The ability to browse the simulation case data in a familiar internet browser-like environment." (HYSYS, 2003)*

Data can be exchanged between two CAPE-OPEN compliant modeling and simulation tools. Such data can be represented in XML, a neutral exchange language. For example, CapeML developed in the European Global CAPE-OPEN project, is a model exchange format in XML-based representation. Examples of process modeling tools that are compliant with CAPE-OPEN standard include HYSYS from Aspen and INDISS (INDISS 2006) from RSI SIMCON. The standard XML file which is used to save the process flow sheets and simulation data in HYSYS can be read in and used directly in the INDISS dynamic simulation environment (INDISS 2006).

2.2 Software Configuration Management

Software configuration management (SCM) is the management process to control the evolution of complex systems. SCM contains the following operational aspects (IEEE 1987):

- *Identification*. The identification operation involves defining the product and the identification of its configuration items. An identification scheme reflects the structure of the product, identifies components and their types, making them unique and accessible in some form.
- *Control*. This operational aspect controls changes to a product and its configuration documentation. Change management and version control are the primary activities involved in control.
- *Status Accounting*. Configuration status accounting provides status and information about a product and its configuration items.

- *Audit and Review*. The audit and review operations verify the consistency of the configuration items against the product.

SCM has been used to track the evolution of complex software systems. SCM enables a trace back to a specific point in time in the evolution history. The concept of a *version* is defined as means to capture time related information.

A version is an entity created at a particular time frame during an evolution of an entity. As the entity evolves, many versions of the entity are created at different time frames. In its application to chemical process modelling, an entity can be a process model. A designer may decide to save the state of the model at a particular point in time. The saved model forms a version of the model entity at that particular time. The different versions of the entity can be used to track the evolution of the model, to support model reuse, or to go back to a previous model state if problems occur in the current model state.

In general, SCM manages two types of space, namely, *product space* and *version space*. The product space outlines the managed entities and relationships of a product, the versioning level, and the storage structure for the managed entities; while the version space defines the entities to be versioned and versioning methods. A product space only contains a single version of a given entity, while the version space contains all versions of a given entity. The product space is often called the *workspace*.

Versioning is concerned with managing all different versions and their relationship. A version is commonly identified by a version identity which is unique in the SCM system. Many SCM systems such as the popular CVS (CVS Website, 2005) use the so-called Dewey notation,

which consists of a pair of numbers, to denote the version identities. For example, the Dewey notation *1.1* represents the first version of an entity, while *1.2* represents the second version, and so on. The first digit is usually reserved for a release of the product. For example, the Dewey notation *2.1* means a new release of the product.

As described above, a version is created by saving a state of the entity at a particular time frame. Subsequently, the saved state (version) can be retrieved for inspection. Two of the essential operations in a SCM system are the saving state operation and the retrieving state operation. The saving state operation involves an action called *Check-in* which stores a snapshot of a given entity, generates a new unique version identity for the entity, and associates the version identity to the saved snapshot. Similarly, the retrieving state operation involves an action named *Check-out* which retrieves a particular snapshot of a given entity if the associated version identity is provided.

2.3 Model Evolution Management using SCM

Among the four operational aspects in SCM, the identification and control aspects are particularly interesting for the management of model evolution in chemical process modelling. Traditional version control and configuration management systems such as Arch, BitKeeper, Clearcase, CVS and Subversion can be applied to modelling of chemical processes to manage the model evolution. Version control systems provide a chemical model designer the capability to trace back to previous versions of individual model files, and to track changes made to the different versions.

In general, a version control system has two control tasks:

1. Provides a mechanism for automatic assignment of unique version identities upon check-in of managed entities
2. Maintains the relationship between different versions.

In order to automate the versioning decision, it is necessary to carry out change detection so as to evaluate the degree of differences between an existing file with its evolved counterpart. This aspect, in a way, is the converse to the work of Surma *et al*, (1996) and Avramenko *et al* (2006) where one of their foci is retrieval based on similarity and as such the issue of similarity measurements are vital for the quality of the retrieved results. Here, the decision as to how to carry out the versioning is based on the quantity of detected changes between two artefacts. Subsequently, the retrieval of a file in a configuration management system (*Check-out* action) is handled by the in-built Access Manager and is based on version attributes that were used in the first instance in evaluating versioning decisions (e.g. version identifier) associated with the same file.

In terms of detecting changes, the Unix *diff* utility, based on the Longest Common Subsequence matching algorithm between two text strings, is probably one of the oldest change detection technique. In terms of detecting changes in XML documents with an underlying tree-structure data model, semantics are associated to different nodes (e.g. “data” in the form of attributes values are stored in the leaves) of the tree and associated change detection techniques are based on tree-differencing algorithms (Shasha *et al*., 1990; Valiente, 2001). Algorithms such as *XyDiff* (Corbéna and Marain,2002) and *X-Diff* (Wang et al, 2003) have been developed to specifically detect changes in XML files.

However, several problems exist with traditional version control tools when they are applied to the management of model evolution in chemical process modeling. These include:

- Version identification is not truly automatic. For example, version identities are automatically provided for the individual configuration items within a version control system such as CVS. These version identities appear as standard revision numbers which are cryptic to a user and which do not contain any information about the nature of the evolution for the configuration items. To overcome this problem, version control systems provide the "tags" mechanism to generate a user-defined version identity for a configuration or a configuration item. However, a user who wants to use this sort of mechanism will have to know the *ins and outs* of the version control tool as to when and how to provide a tagged version identity.
- No tracking of the historical relationships is available. A user of these version control systems has to manually document the evolution trace of the configuration or configuration items in order to find the historical relationships in the evolution. For example, it is difficult to determine whether significant or minor changes have been committed to a particular version of the model based on its revision number as maintained by a version control system.

Furthermore, common version control systems are essentially file-centric, and regard the managed entities as linear text files without considering the hierarchical structures contained in the managed entities. While these version control systems work well for software systems which involve large collections of source codes in various files, this file-centric nature prevents them from being successfully adopted into the management of model evolution in chemical process modelling.

For example, the CapeML models as defined by a CapeML schema/DTD in von Wedel (2002) has the hierarchy structure as shown in Figure 1. For completeness, the CapeML

schema which is modified from the CapeML DTD (von Weldel 2002) is included in appendix. A chemical process model which conforms to the CapeML model structure is a XML instance of the CapeML schema/DTD. Obviously, from a chemical process modelling point of view, adding a new equation changes the model structure, while adding several lines of annotation in the element, *ModelType*, of the CapeML schema/DTD does not have any substantial impact to the model structure of the chemical process.

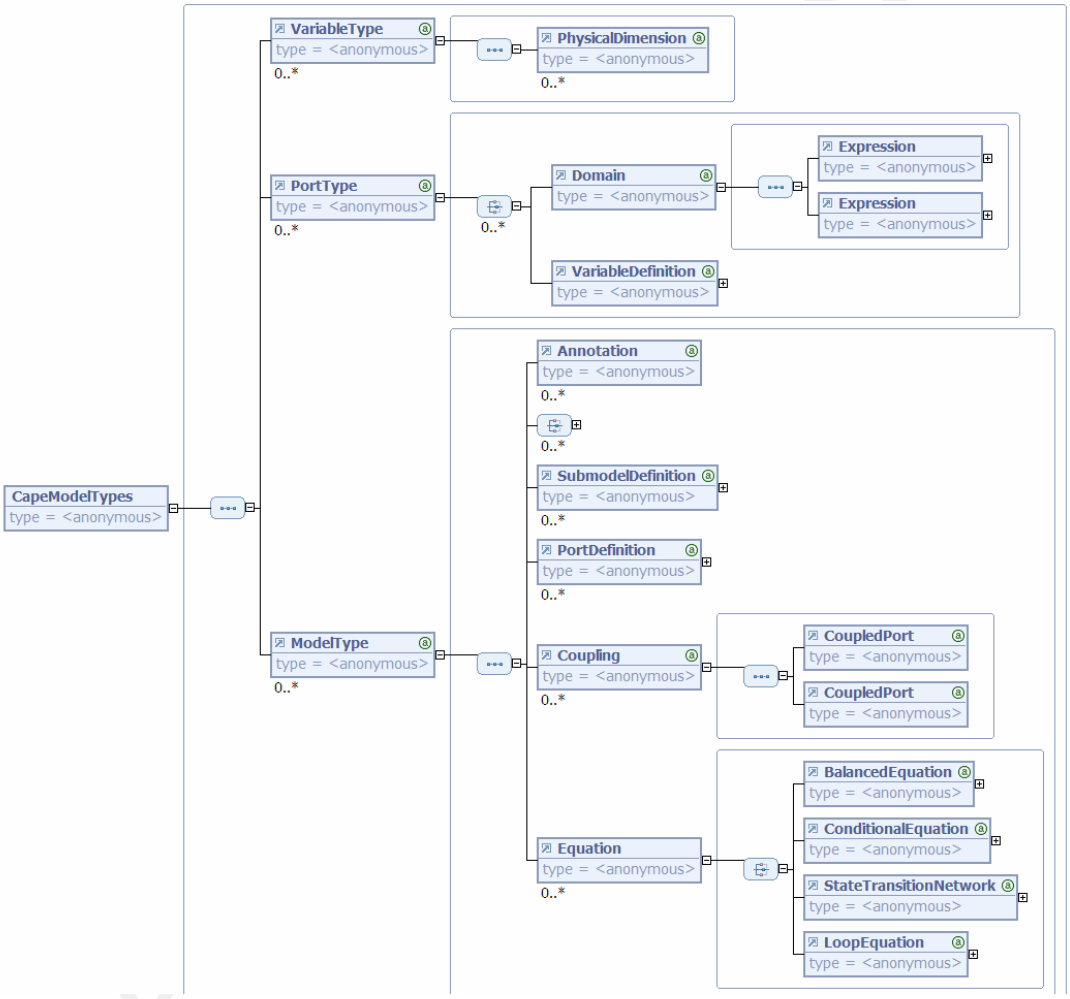

Figure 1 CapeML Model Structure

However, if an ordinary version control system such as CVS is employed to manage the evolution of a chemical process model in CapeML format, the model is only considered as a

linear text file. For an ordinary version control system where changes are detected using differencing algorithms for strings such as the Longest Common Subsequence matching algorithm, adding a new equation which takes 5 lines in the XML instance is the equivalent change in textual lines as to inserting 5 annotation lines in the element type *ModelType*. For CVS and other similar version control tools, both changes will result in the creation of a revision which hides the nature of the change. Clearly, model hierarchy and the evolution of structural information are completely destroyed in this case.

For the management of model evolution in chemical process modelling which conforms to a XML format such as CapeML, two problems need to be considered:

1. *Efficient identification of structural changes between two XML model instances;*
2. *Automatic comprehension of the detected changes with regard to the model structure which leads to automatic version control*

The first problem can be tackled in a straightforward way. Instead of using string-based differencing algorithms to detect changes in model evolution, it is possible to use structure-aware algorithms to detect changes in model evolution. An XML instance is essentially a data model which has a tree structure. Tree differencing algorithms can be applied to detect structural changes between two XML instances, or two trees. For example, the two popular tree differencing algorithms described below can both detect structural changes in XML model instances for chemical processes:

1. *XyDiff*. XYdiff was developed by Corbéna and Marain (2002) to detect changes for ordered XML instances. The XyDiff algorithm can detect changes due to update,

deletion, and insertion of attributes, nodes or sub-trees. Sub-tree moves can also be detected. The XyDiff algorithm has an almost linear runtime complexity.

2. *X-Diff*. X-Diff is developed by *Wang et al.* (2003) to identify changes for unordered XML instances. The X-Diff algorithm can efficiently detect changes due to update, deletion, and insertion of attributes, nodes or sub-trees. This algorithm may not be suitable for XML instances where order is important.

While it may be more appropriate to develop a specific tree differencing algorithm for change detection in XML instances of chemical models as they evolve, existing tree differencing algorithms and tools that detect XML changes is sufficient. We select XyDiff as the tree differencing algorithm in our approach as the source codes for XyDiff are available and are in the public domain.

To the knowledge of the authors, no systematic approach has yet been reported in the literature that has attempted to deal with the second problem discussed above, namely the automatic versioning of chemical models. We propose a model-centric solution to address this problem in the next sub-section.

2.4 Automatic Model-Centric Version Control

It generally takes years and a lot of efforts to develop a version control system from scratch. Many version control systems such as Arch, BitKeeper, Clearcase, CVS and Subversion are stable, and achieve enormous success in a wide range of applications (Better SCM Initiative 2006 and the references therein). Therefore, instead of developing a new version control system from scratch, it may be more desirable to extend an existing mature version control system to form a modified system that addresses the needs of a specific application.

For demonstration purpose, we have selected CVS as the mature version control system that will provide the basic version control functionality associated with this project. The rationale is based on the fact that CVS is a stable and open-source version control system which is arguably one of the most widely used SCM tool in both academic arenas and in industries. Please note that the proposed approach does not depend on the use of CVS as the versioning decision in the approach is independent of the specifics in a SCM tool. Therefore, other mature version control systems such as Arch, BitKeeper, Clearcase and Subversion can be readily used in the place of CVS, although with possibly some differences in terms of the actual implementation.

In our approach, a Model-Centric Version module is added to a CVS system to achieve automatic model-centric version control for managing the evolution of chemical models in XML format. The focus of the approach is on providing an automatic version management mechanism for model reuse. Therefore, it is assumed that the chemical model instances in XML format are automatically generated by modeling tools, (e.g., the model XML files are exported from modeling and simulation tools like HYSYS). For example, one possible scenario is that modellers work in their familiar modeling environment, and manage the modeling differences after a version/revision/modification exercise within the modeling environment with an integrated SCM tool. When necessary, modellers can use the built-in facilities in the modeling environment to export the models in XML format for them to be reused in other modeling and simulation environments. Figure 2 shows the conceptual diagram of the proposed model-centric evolution management system.

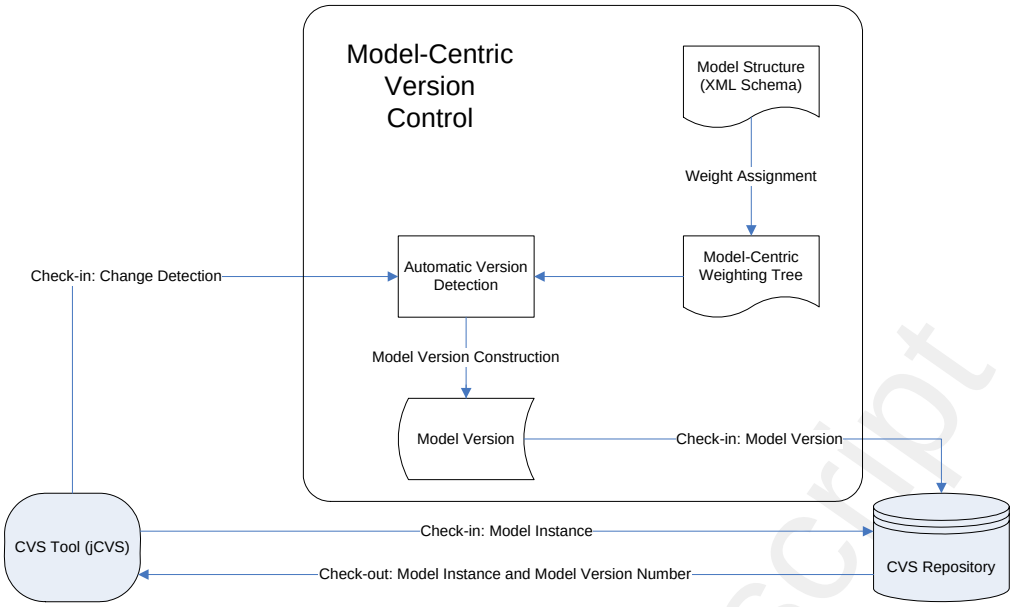

Figure 2 A Model-Centric Evolution Management System

The proposed evolution management system for chemical process modelling consists of a standard CVS system which involves both the CVS server which hosts the CVS repository and the CVS client which provides the user interface of the version control system. The proposed model-centric version control module is integrated with the CVS client. For clarity, in Figure 2, the proposed model-centric version control module is shown with more details.

The proposed model-centric version control module consists three parts:

1. A model-centric weighting tree which is created from the model structure represented by a model schema.
2. Measurement of the changes once they have been detected between a new snapshot of a versioned model and its previous version via the XyDiff algorithm.
3. Automatic version detection using change information corresponding to the model structure and identification of versioning rules.

Once the model version has been automatically determined for the new snapshot, this model version information is used when the corresponding snapshot of the model instance is checked-in the CVS repository. While the CVS server automatically assigns a system-based

revision number using the Dewey notation to the checked-in snapshot, the proposed model-centric version control module will use the information to associate more meaningful tag information about the nature of the model's evolution. As mentioned previously, the model-centric version control module is not CVS dependent and thus for integration with other SCM tools, only the CVS tool (JCVS), CVS repository modules and specific check-in/checkout instructions need to be substituted.

In the following section, the three parts of the model-centric version control modules are discussed in details.

2.4.1 Model-Centric Weighting Tree

As mentioned previously, XML schemas/DTDs can be used as a means of promoting interoperability between chemical process modeling tools. Different tools as well as different chemical process engineers may use different XML schemas. The model-centric weighting tree is used for supporting configuration management of models associated with different schemas as well as to provide a way where individuals may assigned different weights based on their beliefs as to the significance of different parts of a model.

A model structure as defined by a XML schema can be heuristically flattened to form a tree. Different weights are assigned to the nodes of the tree to form a weighting tree for the model structure, as shown in Figure 3. In general, a unique weighting tree can be created for each XML schema types associated process models. In addition, it will also allow different users using the same XML schema to set up their own weighting tree, essentially allowing them to tailor the versioning process (e.g. when are the changes significant enough to be considered a version or a revision).

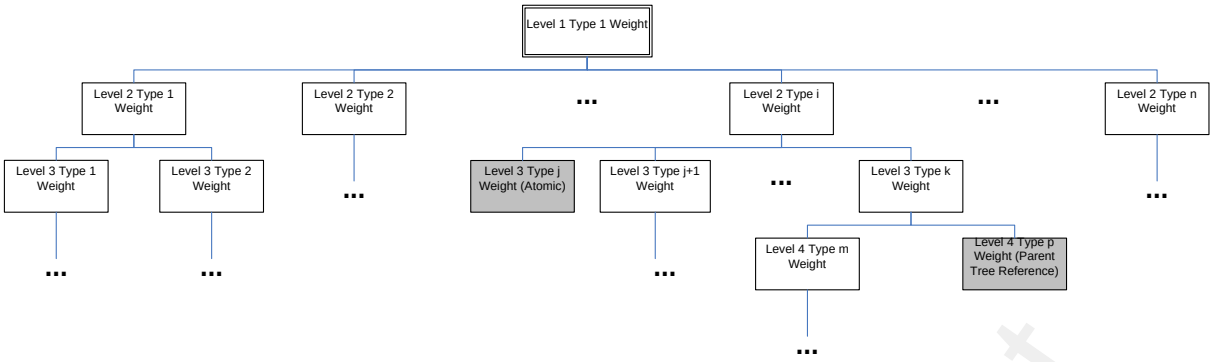

Figure 3 Weighting Tree for a XML Model Structure

The heuristic flattening of a branch in the XML schema tree stops at an atomic node where the element does not have a child element, or at a reference node where the element refer to some elements in its parent branch of the tree. For example, the shadowed Level 3 Type j node in Figure 3 is an atomic node. If the shadowed Level 4 Type p node is a reference node which refers to the element node Level 2 Type i in its parent branch, then no further flattening of the branch is carried out beyond this reference node.

Using the CapeML schema as an example, the associated weight tree can be defined as shown in Figure 4.

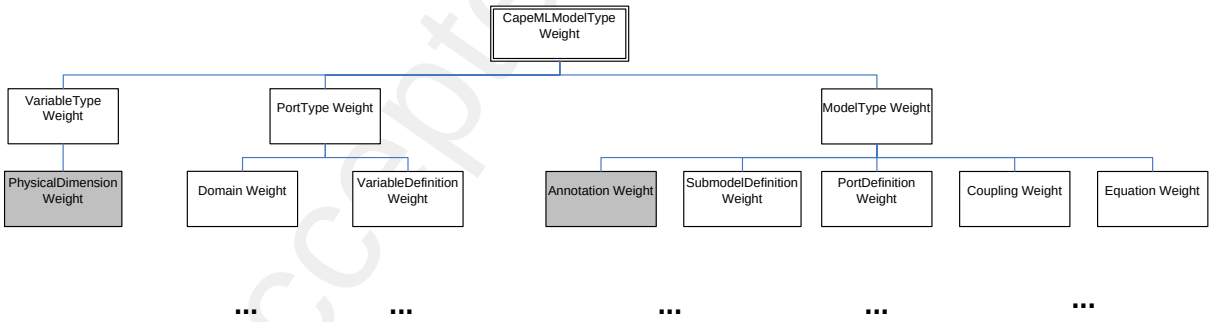

Figure 4 Weighting Tree for the CapeML Model Structure

With the definition of the weighting tree, the model can be classified as a tree structure which contains sub-structures that have different weights assigned to them. A particular sub-structure can be assigned a particular weight to show the importance of that sub-structure in

the view of the chemical model designer. For example, assuming that the default weight for all nodes in the weighting tree for the CapeML model is set to 1, an assignment of a weight of 0.001 to the tag, *Annotation Weigh,* means that extra annotations in the model instances are considered minor for the model designer. Alternatively, an assignment of a weight of 10.00 to the *Equation Weight* confirms that the Equation sub-structure is critical to the model, and any substantial change to the Equation sub-structure should be considered as important since such change may modify the overall model structure for a chemical process.

Once defined, the weighting tree can be mapped to instantiated XML chemical model instances. If changes are made to the XML model instances, these changes can be "measured" using the weighting tree to determine the significance of the changes.

The weighting tree provides a manageable solution to measure the evolution of the chemical process models. A XML schema/DTD defines the fundamental model structure for a class of chemical process models which are XML instances of the XML schema/DTD. Therefore, a weighting tree based on the XML schema/DTD provides a weighting reference for that class of chemical process models. The weighting tree only needs to be generated once for each model class which conforms to a XML schema/DTD. Furthermore, the weighting tree can be generated by a modeling expert who has in-depth knowledge of the model structure represented by the XML schema/DTD. The weighting tree is also transparent to the modeling engineers once it is generated.

In the following section, we use the CapeML model structure and several sample CapeML model instances to demonstrate the determination of the changes. A weighting tree based on the preference of a user or a tool is created for the CapeML model structure. Part of the

weighting tree for the CapeML model type is shown below. All other elements carry the default weight of 1.0.

```
<SCISECUXMLSchemaTree w="0.0" isRef="yes">
 <Annotation w="1.0"/>
 <CapeModelTypes w="1.0">
  <VariableType w="4.0" isRef="yes">
   <PhysicalDimension w="1.0" isRef="yes"/>
  </VariableType>
  <PortType w="4.0" isRef="yes">
    …
  </PortType>
  <ModelType w="1.0" isRef="yes">
   <Annotation w="0.0001" isRef="yes"/>
    …
   <Equation w="10.0" isRef="yes">
    …
```

2.4.2 Measurement of Changes

Before we proceed further, we need to define the Path for an element in a XML instance.

A path for an element in a XML instance is defined here as:

Path=Level 1 Position:Level 2 Position: … :Level n Position

An element at a specific position in a specific level corresponds to a particular element type. The associated path weight for a particular path is:

$$\textbf{Path\_Weight} = \prod_{i=1}^{n} \textbf{\textit{i}th Level Position Weight}$$

It is straightforward to find a path weight for a particular element in a XML instance. For example, the path for the highlight element in the following XML instance will be

Path=1:9:7. As indicated below, the highlighted element is at L3P7 (level 3, position 7) nested within L2P9 (level2, position 9) which is nested within L1P1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
L1P1  <CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
L2P1-2   <VariableType name="Temperature" myID="temperature" lowerBound="0.0"
upperBound="1000.0" unit="K"/>
L2P3-4   <VariableType name="Volume" myID="volume" lowerBound="0.0"
upperBound="1E8" unit="m3"/>
L2P5-6   <VariableType name="Pressure" myID="pressure" upperBound="200" unit="bar"/>
L2P7-8   <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E500" unit="mol"/>
L2P9      <ModelType myID="M-1" name="GasPhase">
L3P1-2              <VariableDefinition myID="V-1" name="T" ref="temperature"/>
L3P3-4              <VariableDefinition myID="V-2" name="p" ref="pressure"/>
L3P5-6              <VariableDefinition myID="V-3" name="V" ref="volume"/>
L3P7-8              <VariableDefinition myID="V-4" name="n" ref="amount_moles" />
L3P9                <Equation>
L4P1                        <BalancedEquation myID="E-1">
L5P1                              <Expression>
L6P1                                    <Term>
L7P1                                          <Factor>
L8P1-2                                              <VariableOccurrence
definition="V-2"/>
L7P2                                          </Factor>
L7P3                                          <Factor mul.op="MUL">
L8P1-2                                              <VariableOccurrence
definition="V-3"/>
L7P4                                          </Factor>
L6P2                                    </Term>
L4P2                              </Expression>
          …
```

and the path weight for the highlighted element (level position weight of specific element is obtained by referring to Figure 4) is

$$\text{Path\_Weight} = \prod_{i=1}^{3} i\text{th Level Position Weight}$$

$$= \text{Level 1 <CapeModelType> Weight} \times \text{Level 2 <ModelType> Weight} \times \text{Level 3 <VariableDefinition> Weight}$$

Note that in the above sample XML model instance, the level and position information in bold face is included for easy comprehension. This information is not part of the XML instance which represents a chemical process model.

In our approach, all changes are detected using the XyDiff algorithm (Corbéna and Marain 2002). Changes to a XML model instance consist of 5 types, namely, updated attributes, deleted attributes, inserted attributes, deleted elements, and inserted elements. The overall formulas for the weight calculation are:

$$\text{Total\_Attributes\_Change\_Weight} = \sum \text{Attribute\_Deletion\_Weight} + \sum \text{Attribute\_Insertion\_Weight} + \sum \text{Attribute\_Update\_Weight}$$

$$\text{Total\_Elements\_Change\_Weight} = \sum \text{Element\_Deletion\_Weight} + \sum \text{Element\_Insertion\_Weight}$$

$$\text{Change\_Weight} = \text{Total\_Attributes\_Change\_Weight} + \text{Total\_Elements\_Change\_Weight}$$

$$\text{XML\_Instance\_Weight} = \sum \text{Element\_Weight}$$

$$\text{Change\_Percentage} = \text{Change\_Weight} / \text{XML\_Instance\_Weight}$$

The following sections provide details of the formulas for the weight calculation for each change type and for the percentage of change.

2.4.2.1 Attribute_Deletion_Weight for a deleted attribute

$$\text{Attribute\_Deletion\_Weight} = \text{Path\_Weight} / (\text{Number of Attributes at the modified element})$$

The Attribute_Deletion_Weight value is calculated using the path_weight and the number of attributes at the modified element. For example, in the XML model segment below, an attribute upperBound="1E8" is deleted at L2P3 which changes the XML model instance from

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
L1P1  <CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
L2P1-2   <VariableType name="Temperature" myID="temperature" lowerBound="0.0"
upperBound="1000.0" unit="K"/>
L2P3      <VariableType name="Volume" myID="volume" lowerBound="0.0" upperBound="1E8"
unit="m3"/>
…
```

to

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
L1P1  <CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
L2P1-2   <VariableType name="Temperature" myID="temperature" lowerBound="0.0"
upperBound="1000.0" unit="K"/>
L2P3      <VariableType name="Volume" myID="volume" lowerBound="0.0"  unit="m3"/>
…
```

From the original XML instance, it can be concluded that the Path to the change is, Path=1:3, and the corresponding Path Weight is:

$$\text{Path\_Weight} = \text{L1 <CapeModelTypes> Weight} \times \text{L2 <VariableType> Weight}$$

$$= 1.0 \times 1.0 = 1.0$$

Since there are 4 attributes at the modified element, Number of Attributes = 4. Therefore, for this attribute deletion,

$$\text{Attribute\_Deletion\_Weight} = 1.0 / 4 = 0.25$$

2.4.2.2 Attribute_Insertion_Weight for an inserted attribute

The weight calculation for inserted attributes is obtained using the modified XML instance. The formula is similar to that for deleted attributes.

$$\text{Attribute\_ Insertion\_Weight} = \text{Path\_Weight} / (\text{Number of Attributes at the modified element})$$

2.4.2.3 Attribute_Update_Weight for an updated attribute

The weight calculation for inserted attributes is obtained using either the original XML instance or the modified XML instance.

$$\text{Attribute\_ Update\_Weight} = 0.5 \times \text{Path\_Weight} / (\text{Number of Attributes at the modified element})$$

2.4.2.4 Element_Deletion_Weight for a deleted element

For a deleted element which may contain children elements, a path to the deleted element is located using the original XML instance. Afterwards, the following pseudo codes are used to calculate the weight for the deleted element:

```
If ( startElement )   // start of the deleted element
   While ( reachEnd = false ) do
        If ( ! hasChildren )   // arrive at a leaf element
                Element_Weight = Path_Weight × Leaf Element Weight;
                Element_Deletion_Weight = Element_Deletion_Weight + Element_Weight;
                Return to parent level;
        Else
                  Traverse into next level;
        Endif;
        If ( endElement )   // arrive at </d>
                reachEnd = true;
        endif;
   endWhile;
endif
```

For example, if lines L7P3, L8P1-2, and L7P4 are deleted from the following model instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
L1P1  <CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
L2P1-2   <VariableType name="Temperature" myID="temperature" lowerBound="0.0"
upperBound="1000.0" unit="K"/>
L2P3-4   <VariableType name="Volume" myID="volume" lowerBound="0.0" upperBound="1E8"
unit="m3"/>
L2P5-6   <VariableType name="Pressure" myID="pressure" upperBound="200" unit="bar"/>
L2P7-8   <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E500" unit="mol"/>
L2P9      <ModelType myID="M-1" name="GasPhase">
L3P1-2              <VariableDefinition myID="V-1" name="T" ref="temperature"/>
L3P3-4              <VariableDefinition myID="V-2" name="p" ref="pressure"/>
L3P5-6              <VariableDefinition myID="V-3" name="V" ref="volume"/>
L3P7-8              <VariableDefinition myID="V-4" name="n" ref="amount_moles" />
L3P9                <Equation>
L4P1                        <BalancedEquation myID="E-1">
L5P1                              <Expression>
L6P1                                    <Term>
L7P1                                          <Factor>
L8P1-2                                              <VariableOccurrence
definition="V-2"/>
L7P2                                          </Factor>
L7P3                                          <Factor mul.op="MUL">
L8P1-2                                              <VariableOccurrence
definition="V-3"/>
L7P4                                          </Factor>
L6P2                                    </Term>
L4P2                              </Expression>
…
```

The paths to each of the deleted elements can be easily obtained

Path=L1P1:L2P9:L3P9:L4P1:L5P1:L6P1:L7P3 <Factor mul.op="MUL">

Path=L1P1:L2P9:L3P9:L4P1:L5P1:L6P1:L7P3:* <VariableOccurrence definition="V-3"/>

</Factor>

In the weight calculation algorithm for the deleted elements, only the leaf element weight will directly contribute to the Element_Deletion_Weight. The parent elements of a leaf element will indirectly contribute to the Element_Deletion_Weight through the Path_Weight.

The path for the leaf element <VariableOccurrence definition="V-3"/> is:

Path=L1P1:L2P9:L3P9:L4P1:L5P1:L6P1:L7P3

And the associated path weight can be obtained as

Path_Weight = <CapeModelTypes> Weight × <ModelType> Weight × <Equation> Weight
× <BalancedEquation> Weight × <Expression> Weight × <Term> Weight
× <Factor> Weight
= 1.0 × 1.0 × 10.0 × 1.0 × 1.0 × 1.0 × 1.0 = 10.0

Therefore,

Element_Deletion_Weight = Path_Weight × Leaf Element Weight
= 10.0 × <VariableOccurrence> Weight = 10.0 × 1.0 = 10.0

2.4.2.5 Element_Insertion_Weight for an inserted element

For an inserted element which may contain child elements, the calculation of the weight is similar to that for a deleted element. However, as the inserted element only appears in the modified XML model instance, the path information can only be obtained from the modified XML instance. Once the path information to the inserted element is obtained, the following pseudo codes are used to calculate the weight for the inserted element:

```
If ( startElement )   // start of the inserted element
   While ( reachEnd = false ) do
        If ( ! hasChildren )   // arrive at a leaf element
              Element_Weight = Path_Weight × Leaf Element Weight;
              Element_ Insertion _Weight = Element_ Insertion _Weight + Element_Weight;
              Return to parent level;
        Else
                Traverse into next level;
        Endif;
        If ( endElement )   // arrive at </i>
              reachEnd = true;
        endif;
   endWhile;
endif
```

2.4.2.6 XML_Instance_Weight for the original XML instance

The pseudo codes to calculate the instance weight for the original XML model instance is shown below:

```
While ( reachEnd = false ) do  // start from the top level
        If ( ! hasChildren )   // arrive at a leaf element
                Element_Weight = Path_Weight × Leaf Element Weight;
                XML_Instance_Weight = XML_Instance_Weight + Element_Weight;
                Return to parent level;
        Else
                  Traverse into next level;
        Endif;
        If ( endElement )   // arrive at the end of the top level
                reachEnd = true;
        endif;
endWhile
```

2.4.3 Automatic Version Detection

The following version identifiers are adopted here for the management of chemical model evolution:

- *New Version*. A new version represents a significant change in the evolution of a chemical process model. For example, if the model structure is changed significantly, a new version is created.
- *Revision*. A revision commonly represents substantial changes in the evolution of the chemical process model. However, such changes may not alter the model structure. For example, several constants of the model are modified to tailor to a new simulation while the overall model structure is not changed
- *Minor Change*. A minor change represents small changes to the model. For example, the lower bound of a parameter is modified which should be classified as a minor change for the model.

Contrary to the Dewey notation used in CVS revision, we define our Version Identity (VID) as:

VID = T-Release-New_Version-Revision-Minor_Change

Therefore, a VID of T-1-1-2-1 identifies a snapshot of a model which is the first version of the first release. If another snapshot for the same model has a VID of T-1-1-2-2, then the new snapshot has minor change over the T-1-1-2-1 snapshot. If the third snapshot for the model has a VID of T-1-1-3-2, this indicates that substantial changes i.e. revision) have been made to the model. A VID of T-1-2-0-0 simply means that the model structure has been changed (i.e. a new version has been created).

The automatic version detection is achieved using three thresholds T_1, T_2 and T_3 which are relevant to the model structure in use. For given real number T_1, T_2 and T_3

$$0 < T_1 < T_2 < T_3$$

Then version rules used are:

- **New Version**: Change_Percentage $\geq T_3$
- **Revision**: $T_2 \leq$ Change_Percentage $< T_3$
- **Minor Change**: $T_1 \leq$ Change_Percentage $< T_2$
- **No Change**: $0 \leq$ Change_Percentage $< T_1$

Note that *No Change* in the above rules means that there is no change to the model, or the only changes are documentation in the form of annotation and thus considered to be

insignificant. While documentation may help a model designer to comprehend a model, it is not the essential part of the model. Therefore, documentation-only changes are classified as *No Change* in the proposed approach.

Clearly, the three thresholds T_1, T_2 and T_3 have important impacts on the automatic version detection. Unfortunately, there is no fix selection criterion for the three thresholds as they are closely related to the particular model structure under consideration, further research is needed in order to establish guidelines for the selection of the thresholds.

Before we proceed to the tool support section, it is necessary to present a brief discussion about the measurement for the model evolution.

It is likely that the measurement of the model evolution can be affected by the inconsistency introduced in the modeling process. For example, a Werder equation

$$\log(P) = a + b/T$$

may be alternatively modeled by introducing a new substitution variable c manually:

$$\log(P) = a + c$$

$$c=b/T$$

While the alternative model is equivalent to the original one, the difference between the two models may be significant using the proposed measurement.

However, with the various modeling tools available to assist the modeling process, the inconsistency such as the one shown above is arguably caused by manual modeling practice only. Different modeling tools use different approaches to model chemical processes. The most popular tools are: 1) flowsheeting modeling tools such as Aspen+ or HYSYS which use unit operation modules where the simulation models are assembled from predefined libraries of unit operation modules; 2) block modeling tools such as Matlab Simulink in which the process simulation models are constructed using input-output blocks from block libraries; and 3) equation modeling tools such as gPROMS in which the process simulation models are represented as mathematical equations. In general, it is difficult to introduce the modeling inconsistency such as the one shown above in flowsheeting modeling tools and block modeling tools as the modules and blocks are usually predefined. The introduction of the trivial variations is possible in equation modeling tools; however, such inconsistency may cause unnecessary overhead in simulation and thus should be avoided. In this sense, a significant difference may indicate to the modeler that potential modeling inconsistency exists in the evolution of the process model.

As the proposed method is designed to measure the model evolution process where the model XML instances are automatically generated by modeling tools, it is likely such modeling inconsistency can be effectively avoided. If a new variable c is introduced in the modeling tool so that the exported XML instance takes the form of the alternative model, and if the variable c is not trivial, then such introduction of a non-trivial process variable should represent significant change in the model.

Alternative models may also be created for simulation purposes. For example, a developed model can be further tuned to accommodate variations in simulation environments such as introduction of alternative numeric solvers. Traditionally, such model tuning in chemical

process modeling is often accomplished by creating an alternative working folder to host the alternative model for the new numeric solver. Such practice is better managed by using the 'branch' feature which is available in almost every modern version control systems. A branch in a version control system is like a new working folder in file-based modeling environment to facilitate parallel development of modeling activities such as model tuning for different computational units. Unfortunately, it is unlikely that an automatic decision can be made with regard to branching in most version control systems for such a situation, as when to branch is mostly a management decision. The modeler may issue the instruction to the version control system for branch creation and once a branch is created the proposed method can be applied to provide automatic decision for revision control of the new branch.

3. Verification of Approach Using A Prototype Tool and A Case Study

A prototype version control system that implemented the proposed model-centric evolution management approach described in Section 2 has been developed. In this prototype tool the model-centric version control module is integrated with the open source jCVS tool (jCVS Website, 2005). The weighting tool is first described and then using a case study, six usage scenarios that demonstrates the use of the developed approach in the management of the evolution of a sample chemical model is shown in the subsequent sections.

As shown in Figure 5, a weighting tool has been implemented in the proposed model centric version control system. This weighting tool allows a user to input a user-specific XML schema associated with a chemical model and automatically performs the heuristic flattening associated with the model structure to create a tree structure. This tree structure is displayed on the panel on the left in the user interface shown in Figure 5. A user is able to mouse-click

on a specific element within this panel to change its corresponding weight. For ease of use, all weights were initially set to a default value and users can chose to change the weights of elements with specific relevance to them using the panel on the right. Thus, based on process domain knowledge, a user can assign different weights to various nodes of the tree structure, and a weighting tree will be automatically generated for the model schema. The user also has the option to reset the weights to the default values as well as to export the changes in a XML file. As an example, figure 6 shows an automatically generated weighting tree for the CapeML schema. Note that in most cases, the weights associated with a XML schema for a chemical model needs to be set up only once at the start and will be used throughout its lifespan. The weighting tree tool can also read in an existing XML instance of the weighting tree. A user can then modify the weight assignments and save a modified XML instance of the weighting tree.

The user interface of the model-centric version control system is simplified from the user interface used by jCVS (jCVS Website, 2005). Figure 7 shows the check-out interface of the system.

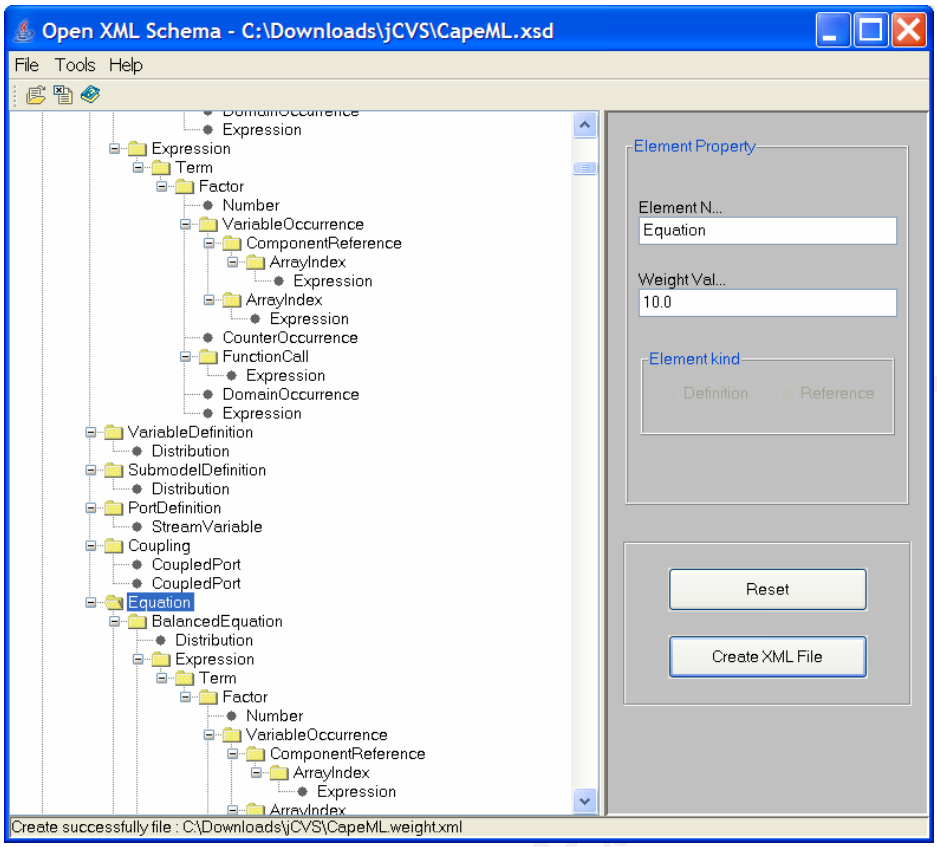

Figure 5 Weighting Tree Tool in the Version Control System

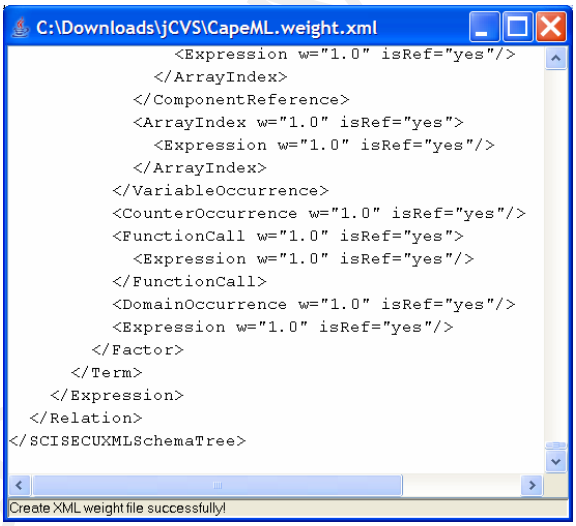

Figure 6 A Weighting Tree for CapeML

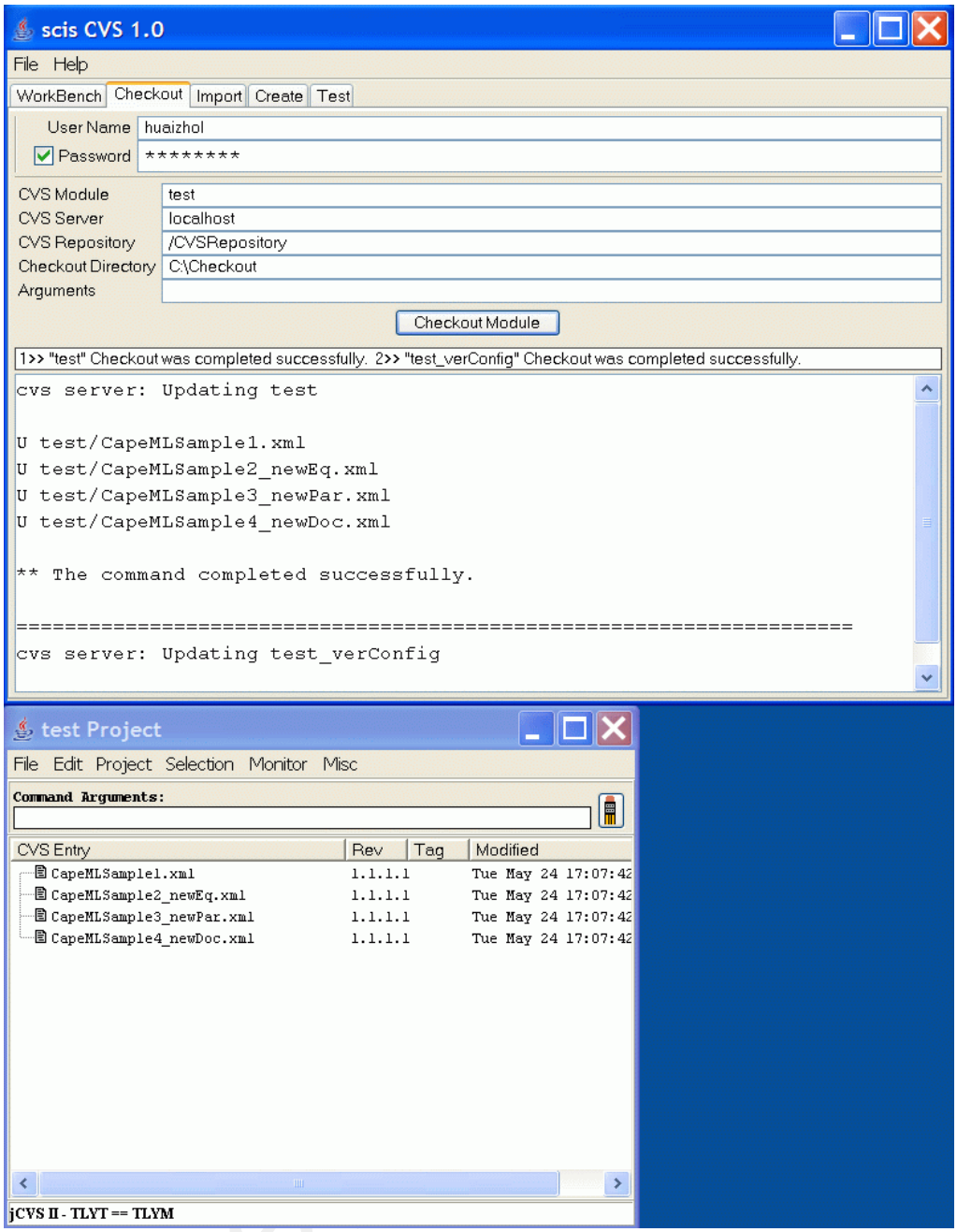

Figure 7 A User Interface of the Model-Centric Version Control System

In the following section, we use a case study to demonstrate the use of the proposed tool in the management of the evolution of a sample chemical model and the associated automatic version control. For ease of discussion we continue to use the CapeML schema as the example schema. A sample model instance, namely, *CapeMLSample1.xml* which conforms to the CapeML schema is used for the evolution purpose. The weighting tree for the CapeML schema has already been described in Section 2.

3.1 Case Study

We first provide the thresholds used in the case study:

```
<T>
 <T1>0.001</T1>
 <T2>0.05</T2>
 <T3>0.2</T3>
</T>
```

The starting point involves the initial checked-out XML instance model as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
	<VariableType name="Temperature" myID="temperature" lowerBound="0.0" upperBound="1000.0"
unit="K"/>
	<VariableType name="Volume" myID="volume" lowerBound="0.0" upperBound="1E8" unit="m3"/>
	<VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="200" unit="bar"/>
	<VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E500" unit="mol"/>
	<ModelType myID="M-1" name="GasPhase">
		<VariableDefinition myID="V-1" name="T" ref="temperature"/>
		<VariableDefinition myID="V-2" name="p" ref="pressure"/>
		<VariableDefinition myID="V-3" name="V" ref="volume"/>
		<VariableDefinition myID="V-4" name="n" ref="amount_moles" />
		<Equation>
			<BalancedEquation myID="E-1">
				<Expression>
					<Term>
						<Factor>
							<VariableOccurrence definition="V-2"/>
						</Factor>
						<Factor mul.op="MUL">
							<VariableOccurrence definition="V-3"/>
						</Factor>
					</Term>
				</Expression>
				<Expression>
					<Term>
						<Factor>
							<VariableOccurrence definition="V-1"/>
						</Factor>
						<Factor mul.op="MUL">
							<VariableOccurrence definition="V-4"/>
						</Factor>
						<Factor mul.op="MUL">
							<Number value="8.3144"/>
						</Factor>
					</Term>
				</Expression>
			</BalancedEquation>
		</Equation>
	</ModelType>
</CapeModelTypes>
```

In order to demonstrate how the proposed approach manages the various types of modification in the evolution of a model, various scenarios involving this XML model instance are shown in this section. A point to note is that although the XML model instance is used here in each scenarios to demonstrate the modifications made to the model, in practice the modeller makes these changes in the modelling environment (e.g. HYSYS) and when he/she selects the save option, the XML model instance is generated by the system automatically and is subsequently used by the model-centric version control module to make a versioning decision. The XML format is internal to the system and everything that is carried out by the modeller is within the modelling environment.

Scenario 1: Inserted Comments into Model

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temperature" myID="temperature" lowerBound="0.0" upperBound="1000.0"
unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.0" upperBound="1E8" unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="200"
unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E500" unit="mol"/>
        <ModelType myID="M-1" name="GasPhase">
                <Annotation comment="This is a GasPhase model" ref="M-1"/>
                <VariableDefinition myID="V-1" name="T" ref="temperature"/>
                …
```

As shown in the highlighted section in the figure above, a comment inserted by the modeller in the modelling environment has been added to the element, *ModelType* in the XML model instance (shown as the *Annotation* element). Note that comments do not change the model structure or the technical content at all. Therefore, via a dialog box, the version control system reports *No Change* to this modification when this modified instance is checked in.

Scenario 2: Modifications involving a single attribute

In this scenario the modeller has made a change to one of the attributes, namely, the value associated with '*upperBound*' for the variable '*Pressure*' (highlighted in the figure below). According to our version definitions, this should represent only a minor change to the existing model.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temperature" myID="temperature" lowerBound="0.0" upperBound="1000.0"
unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.0" upperBound="1E8" unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="201"
unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E500" unit="mol"/>
        <ModelType myID="M-1" name="GasPhase">
                <Annotation comment="This is a GasPhase model" ref="M-1"/>
                …
```

Minor Change is reported via a dialog box when this modification is checked into the configuration management system. The version identity for this modified file is changed to T-1-1-0-1.

Scenario 3: Modifications involving multiple attributes

The modeller makes modifications to a number of attributes associated with the variables 'Temperature', 'Volume' and 'AmountMoles' respectively. As these modifications do not change the model structure, they may be regarded as minor changes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temp" myID="temp" lowerBound="0.0" upperBound="1000.0" unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.1" upperBound="1E8"
unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="201"
unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E300" unit="kmol"/>
        <ModelType myID="M-1" name="GasPhase">
                <Annotation comment="This is a GasPhase model" ref="M-1"/>
                …
```

Similarly, Minor Change is reported for this scenario from the prototype system and the VID is changed accordingly upon check in of the file.

Scenario 4: A new variable is introduced, but this variable is not used in the model equation structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temp" myID="temp" lowerBound="0.0" upperBound="1000.0" unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.1" upperBound="1E8"
unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="201"
unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E300" unit="kmol"/>
        <VariableType name="Weight" myID="weight" lowerBound="0.0" upperBound="1E1000"
unit="kg"/>
        <ModelType myID="M-1" name="GasPhase">
                <Annotation comment="This is a GasPhase model" ref="M-1"/>
                <VariableDefinition myID="V-1" name="T" ref="temp"/>
                <VariableDefinition myID="V-2" name="p" ref="pressure"/>
                <VariableDefinition myID="V-3" name="V" ref="volume"/>
                <VariableDefinition myID="V-4" name="n" ref="amount_moles" />
                <VariableDefinition myID="V-5" name="W" ref="weight" />
                <Equation>
                …
```

Upon check-in, the version control systems reports a revision for this change and the VID is changed to T-1-1-1-2. We believe that a revision is an appropriate decision for this scenario as only a new variable is introduced.

Scenario 5: The introduced new variable is added into an expression in the model equation structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temp" myID="temp" lowerBound="0.0" upperBound="1000.0" unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.1" upperBound="1E8" unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="201" unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E300" unit="kmol"/>
        <VariableType name="Weight" myID="weight" lowerBound="0.0" upperBound="1E1000" unit="kg"/>
        <ModelType myID="M-1" name="GasPhase">
          <Annotation comment="This is a GasPhase model" ref="M-1"/>
                <VariableDefinition myID="V-1" name="T" ref="temp"/>
                <VariableDefinition myID="V-2" name="p" ref="pressure"/>
                <VariableDefinition myID="V-3" name="V" ref="volume"/>
                <VariableDefinition myID="V-4" name="n" ref="amount_moles" />
                <VariableDefinition myID="V-5" name="W" ref="weight" />
                <Equation>
                        <BalancedEquation myID="E-1">
                                <Expression>
                                        <Term>
                                                <Factor>
                                                        <VariableOccurrence definition="V-2"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <VariableOccurrence definition="V-3"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <VariableOccurrence definition="V-5"/>
                                                </Factor>
                                        </Term>
                                </Expression>
                                …
```

However, it can be observed from the XML instance that the change to the model structure is not significant. The version control system therefore reports a revision for this scenario.

Scenario 6: The model structure is changed significantly. An expression is removed from the model equation.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file -->
<CapeModelTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Schema\CapeML.xsd">
        <VariableType name="Temp" myID="temp" lowerBound="0.0" upperBound="1000.0" unit="K"/>
        <VariableType name="Volume" myID="volume" lowerBound="0.1" upperBound="1E8" unit="m3"/>
        <VariableType name="Pressure" myID="pressure" lowerBound="0.0" upperBound="201"
unit="bar"/>
        <VariableType name="AmountMoles" myID="amount_moles" lowerBound="0.0"
upperBound="1E300" unit="kmol"/>
        <VariableType name="Weight" myID="weight" lowerBound="0.0" upperBound="1E1000"
unit="kg"/>
        <ModelType myID="M-1" name="GasPhase">
          <Annotation comment="This is a GasPhase model" ref="M-1"/>
                <VariableDefinition myID="V-1" name="T" ref="temp"/>
                <VariableDefinition myID="V-2" name="p" ref="pressure"/>
                <VariableDefinition myID="V-3" name="V" ref="volume"/>
                <VariableDefinition myID="V-4" name="n" ref="amount_moles" />
                <VariableDefinition myID="V-5" name="W" ref="weight" />
                <Equation>
                        <BalancedEquation myID="E-1">
                                <Expression>
                                        <Term>
                                                <Factor>
                                                        <VariableOccurrence definition="V-2"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <VariableOccurrence definition="V-3"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <VariableOccurrence definition="V-5"/>
                                                </Factor>
                                        </Term>
                                </Expression>
                                <Expression>
                                        <Term>
                                                <Factor>
                                                        <VariableOccurrence definition="V-1"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <VariableOccurrence definition="V-4"/>
                                                </Factor>
                                                <Factor mul.op="MUL">
                                                        <Number value="8.3144"/>
                                                </Factor>
                                        </Term>
                                </Expression>
                        </BalancedEquation>
                </Equation>
        </ModelType>
</CapeModelTypes>
```

As the model structure is changed significantly, this modification should result in a new version. Indeed, a new version is created for this scenario, and a VID of T-1-2-0-0 is assigned.

For the above scenarios, the CVS server maintains its own revision numbers in the corresponding CVS repository for the model. There is a clear relationship between the CVS revision numbers and the model version identities assigned by the proposed model-centric version control system, as shown in the table below. This relationship can be used subsequently by the proposed approach to formulate the request to retrieve a given model version from the CVS repository.

Table 1 Relationship between CVS Revision Numbers and Model Version Identities

CVS_Revision	Model_Version	Changes
1.1	T-1-1-0-0	Null
1.2	T-1-1-0-0	NOCHANGE
1.3	T-1-1-0-1	MINOR_CHANGE
1.4	T-1-1-0-2	MINOR_CHANGE
1.5	T-1-1-1-2	REVISION
1.6	T-1-1-2-2	REVISION
1.7	T-1-2-0-0	NEW_VERSION

4. Discussion

Three areas in the proposed approach warrant a discussion and will require further research. The first area relates to the selection of the use of *XyDiff* which then subsequently drives how we go about the weight evaluation for the different types of changes. While the approach is adequate, XyDiff is designed to detect changes in XML files, specifically XML web-based log files and thus is not the most ideal algorithm for detecting changes in very structured XML chemical model files. What would be more ideal is the development of a XML-based differencing algorithm that will be able to carry out semantic differencing by making use of the structure and nomenclature normally associated with chemical models. Using the results of semantic differencing can aid in the development of a more concise approach in terms of the weight evaluation for the different types of changes. Although the proposed approach

implicitly uses structural information, the weight evaluation process is quite complex and tedious.

The automatic model centric versioning component uses three thresholds, the model-centric weighting tree and some rules. An important aspect in this approach is consistency in the versioning decisions – for the same type of changes; the same versioning decision should be obtained. This is achieved via the model-centric weighting tree and the threshold values. The model-centric weighting tree of an XML schema for a class of chemical models captures the rationale of a modeller from the perspective of the significance of changes made to different parts of the model by the assignment of different weightings. This is done once and is used subsequently against ALL changed XML model instances to determine versioning information, thus allowing for consistency. While the weights assigned MUST be appropriate (from a chemical modelling aspect) and reflective of the importance of different elements in the model (e.g. weight associated with a change to an equation block being higher than a change in the value of a parameter), the exact preciseness of these values (for example, a weight value of 1.2 instead of 1.0 being used) is not super-critical in the approach because they are used to aid the management of versioning the chemical models. These values play no part in the correctness/validity of the actual chemical models as their correctness depends on the modelling and not on how the models are saved and stored subsequently.

As mentioned previously, there is no definitive criterion to use in terms of fixing the values of the three thresholds since they are linked strongly to the structure of the models. The only rule of thumb used is that higher weights should be associated with the more significant elements in a model and that changes to more significant elements (possibly leading to structural change in the model) should lead to version rather than a revision at the point of checking into the model management system. Thus at present the structure of the model, the weighting tree

provided by the user and the above rule of thumb is used to determine these three values empirically. In terms of selecting "correct" threshold values for versioning, changes to these, for example in $T_{2,}$ may result in a model being saved and stored as one category (i.e. a revision) rather than another (i.e. a Minor change). However, as the same set of information is used for retrieval, it will not impact the retrieved results. In the above example, the system will then retrieve the file as a revision rather than a Minor change. As an analogy, there is a filing cabinet with cabinets labelled A, B, and C. Items are to be sorted and placed into each of these 3 cabinets. If the rules used to retrieve the item subsequently are the same as those used to decide where the item is to be placed in the first place, the exactness of the rules may not be super critical. If a change in value for one of the rules (e.g. a small change in the threshold value) actually changes the location of where the item is placed, the item will still be consistently retrieved if the same set of information is used in the retrieval.

However, a future addition to the system to address the issue of determining these threshold values is to incorporate an incremental learning module which will learn over time the preference of a user in terms of the categories of versioning. The system will be set up as before but each time a decision has been made, the user will also be asked for their preference as well. The system will then learn to adapt to the user's preference by using the user's input to adjust the values of these thresholds via an incremental machine learning algorithm. The learned values can then be incorporated subsequently for versioning decisions and retrieval.

The last area involves the development of a more sophisticated approach to aid better re-use of chemical models. While the proposed approach provides more information in terms of the changes and that versioning is on the basis of the types of change, the retrieval of detailed information associated with the rational is still a tedious process. Future work here is linked

with the development of the semantic differencing algorithm mention above as such an algorithm will enable the creation of a semantic delta that can be used directly to provide the required information.

5 Conclusion

The paper proposed an automatic model centric version control approach for managing the evolution of chemical models. The use XML-based schemas/DTDs for representing chemical models in terms of supporting interoperability between tools have emerged as one of the many outcomes from the CAPE-OPEN project. Given that chemical modelling tools like HYSYS has just delivered an XML infrastructure that aims to support collaborative engineering , this proposed technique is timely and relevant.

A prototype tool incorporating the proposed approach and the use of XML-based representations of chemical models was implemented as a proof of concept. One of the aims in this project was that the developed approach should support different users with their individually defined XML-based schemas/DTDs. The weighting tool allows users to input any existing valid XML schemas and generates the associated XML instance of the weighting tree. Flexibility in terms of versioning is another area that this project tries to address. By allowing the different users to define weights associated with the various components that makes up a chemical model is an approach that supports user-defined versioning (i.e. the users are given a way of inputs to signal when changes are significant enough to them to warrant, for example, a new version rather than a revision to be saved). Subsequently, different users using their process domain knowledge can modified the weights of the elements in the weighting tree and then to use the tool to manage the evolution of their chemical models. In addition, the approach uses structural changes of the chemical models as the basis for versioning. The prototype tool, as a proof of concept, supports automatic storage

and retrieval of the chemical models as they evolve and control changes made to these models as well as support chemical model reuse.

Acknowledgements

The authors acknowledge the financial and in-kind support provided by the industry partner, Daesim Technologies Pty Ltd, Australian Research Council and Edith Cowan University, without which the project would not have been possible. The authors would also like to thank the anonymous reviewers for their valuable comments in terms of improving the paper.

Reference

Avramenko, Y., Kraslawski, A., (2006), Similarity Concept for Case-based Design in Process Engineering. Computers & Chemical Engineering, Vol. 30, pp. 548-557.

Bayer, B., and Marquardt, W. (2003). A Comparison of Data Models in Chemical Engineering. Concurrent Engineering Research and Applications, Vol. 11, No. 2, June 2003, pp.129-138.

Bayer, B., and Marquardt, W. (2004). Towards integrated information models for data and documents. Computers and Chemical Engineering, Vol. 28, 2004, pp. 1249-1266.

Better SCM Initiative (2006). http://better-scm.berlios.de/ Access in November 2006.

Bogusch, R., Lohamnn, B., and Marquardt, W. (1996). Computer-aided Process Modelling with MODKIT. Chemputers Europe III, Frankfurt, October 1996.

Caballero, C. (1994), Life Cycle: Now the focus in Unix CM Market, Application Development Trends. (64)(86), pp. 49-54.

Corbéna, G. and Marain, A. (2002). Detecting Changes in XML Documents. Proc. IDCE 2002, San Jose CA.

CVS Website (2005). http://www.nongnu.org/cvs/ Access in December 2005.

Department of Defense. (1985), Defense Systems Software Development. United States Department of Defense, DOD-STD 2167.

Foss, B. A., Lohmann, B., and Marquardt, W.(1998) A field study of the industrial modelling process. Journal of Process Control.

Goodwin, R., and Chung, P. (1994), An intelligent information system for design, In The European Symposium on Computer-aided process Engineering (ESCAPE 4), pp.375-382.

Heller, M., Schleicher, A. and Westfechtel, B (2004). Process Evolution Support in the AHEAD System, Proc. of 2nd International Workshop Applications of Graph Transformation with Industrial Relevance (AGTIVE'03), Charlottesville, USA, 2003, Revised Selected and Invited Papers, LNCS 3062, pp. 454-460, Springer Verlag, (2004)

HYSYS (2003). HYSYS 3.2 User Guide. Aspen Technology.

IEEE (1987). IEEE Guide to Software Configuration Management. IEEE/ANSI Standard 1042-1987.

INDISS (2006). INDISS Simulation Platform. RSI SIMCON. http://www.simulationrsi.net/RSI-products_services-indiss.htm Accessed in October, 2006.

jCVS Website (2005). http://www.jcvs.org/ Access in December 2005.

Jarke, M., and Marquardt, W. (1996), Design and Evaluation of Computer-Aided Process Modelling Tools, In J.F. Davis, G. Stephanopoulos, V. Venkatasubramanian (Eds.): "Intelligent Systems in Process Engineering", AICHE Symp. Ser.312, Vol.92, pp 97-109.

Jarke, M., Köller, J., Marquardt, W., Wedel, L. v., Braunschweig, B., (1999) CAPE-OPEN: Experiences from a Standardization Effort in Chemical Industries, SIIT'99.

Karhela, T. (2002). A Software Architecture for Configuration and Usage of Process Simulation Models: Software Component Technology and XML-based Approach. PhD Thesis, VTT Technical Research Centre, Finland

Kondelin, K., Karhela, T., Laakso, P. (2004). Service Framework Specification for Process Plant Lifecycle, VTT Research Notes 2277.

Larsson, J., Johansson, B., Krus P. and Sethson M. (2002). Modelith: A Framework Enabling Tool-Indendent Modelling and Simulation, Proceedings of the European Simulation Symposium 2002, Dresden, Germany.

Li, H. and Lam, C. P. (2004). An Exchange Language for Process Modelling and Model Management. Proc. 7th International Symposium on Dynamics and Control of Process Systems, Boston, US.

Marquardt, W., von Wedel, L. and Bayer, B. (2000). Perspectives on lifecycle process modelling. In: Foundations of computer-aided process design (M.F. Malone, J.A. Trainham and B. Carnahan, Eds.) Vol. 96. pp. 192--214. AIChE Symp. Series 323.

Marquardt, W. and Nagl, M. (2004). Workflow and information centered support of design processes — the IMPROVE perspective, Computers & Chemical Engineering, vol. 29, no. 1, pp.65-82 (2004)

Modelith (2006). Modelith: An XML-Based Model Representation for Transformation and Exchange. http://hydra.ikp.liu.se/modelith/. Accessed in October, 2006.

Pantelides, C. C., and Britt, H.I. (1994), Multipurpose process modelling environments, In Foundations Computer-Aided Design Conference FOCAPD'94. , CACHE, Snowmass, CO, pp 128-141.

Ponton, J. (1995), Process Systems Engineering: Halfway through the first century, Chemical Engineering Science, 50(24), 4045-4059.

Schopfer, G., von Wedel, L., and Marquardt, W. (2000). An Environment Architecture to Support Modelling and Simulation in the Process Design Lifecycle. Proceedings of AIChe Annual Meeting 2000, Los Angeles.

Schuchardt, K., Oluwole, O., Pitz, W., Rahn, L.A., Green,Jr., W.H., Leahy, D., Pancerella, C., Sjoberg, M., and Dec, J. (2005). New Approaches for Collaborative Sharing of Chemical Model Data and Analysis Tools. 2005 Joint Meeting of the U.S. Sections of The Combustion Institute, Philadelphia, PA, United States. Technical Report UCRL-CONF-209117, Lawrence Livermore National Laboratory.

Seuranen, T., Karhela, T., and Hurme, M. (2005). Automated Process Design Using Web-Service based Parameterised Constructors. Computer-Aided Chemical Engineering, Vol. 20, 2005, pp. 1639-1645.

Shasha, D. and Zhang, K. (1990). Fast algorithms for the unit cost editing distance between trees. Journal of Algorithms, Vol. 11, No. 4, 1990, pp. 581-621.

Subrahmanian, E., Konda, S., Levy, S., Monarch, I., Reich, Y., and Westerburg, A. (1993). Computational support for shared memory in design. In A. Tzonis and I. White (Eds) "Automation Based Creative Design: Current Issues in Computers and Architecture, Elsevier.

Surma, J., Braunschweig, B., (1996a), REPRO: Supporting Flowsheet Design by Case-Based Retrieval, Proceedings of the Third European Workshop on Case-Based Reasoning: EWCBR'96, pp.400-412.

Surma, J.; Braunschweig, B., (1996b), Case-based retrieval in process engineering: supporting design by reusing flowsheets. Application of Artificial Intelligence, 9(4), 385-391.

Wang, Y., DeWitt, D. J., and Cai, J.-Y. (2003). X-Diff: An effective change detection algorithm forXML documents. Proc. 19th Int. Conf. Data Engineering. Bangalore, India.

Valiente, G. (2001), An efficient bottom-up distance between trees. Proceedings of the 8th International Symposium on String Processing and Information Retrieval, 2001.

von Wedel, L., Marquardt, W. (2000). ROME: A Repository to Support the Integration of Models over the Lifecycle of Model-based Engineering Processes, In: S. Pierucci (Ed.): European Symposium on Computer Aided Process Engineering-10, Elsevier, 2000, 535-540.

von Wedel, L. (2002). CapeML - A Model Exchenge Language for Chemical Process Modelling. Aachen University, Germany.

von Wedel, L. (2003). An Environment for Heterogeneous Model Management in Chemical Process Engineering. PhD Thesis. Aachen University, Germany.

Westerberg, A. W., Subrahmanian, E., Reich, Y., Konda, S. (1997), and the n-dim group. Designing the Process Design Process. Computers Chem. Engg., Vol.21, Suppl., pp S1-S9.

Appendix CapeML Schema

The CapeML schema in the following is modified from the CapeML DTD in von Wedel (2002):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="Annotation">
		<xs:complexType>
			<xs:complexContent>
				<xs:restriction base="xs:anyType">
					<xs:attribute name="comment" type="xs:string" use="required"/>
					<xs:attribute name="ref" type="xs:IDREF" use="required"/>
				</xs:restriction>
			</xs:complexContent>
		</xs:complexType>
	</xs:element>
	<xs:element name="CapeModelTypes">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="VariableType" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="PortType" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="ModelType" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="VariableType">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="PhysicalDimension" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="myID" type="xs:ID" use="required"/>
			<xs:attribute name="math.type" default="REAL">
				<xs:simpleType>
					<xs:restriction base="xs:NMTOKEN">
						<xs:enumeration value="REAL"/>
						<xs:enumeration value="BOOLEAN"/>
						<xs:enumeration value="INTEGER"/>
					</xs:restriction>
				</xs:simpleType>
			</xs:attribute>
			<xs:attribute name="lowerBound" type="xs:string" use="required"/>
			<xs:attribute name="upperBound" type="xs:string" use="required"/>
			<xs:attribute name="std.value" type="xs:string"/>
			<xs:attribute name="unit" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="PhysicalDimension">
		<xs:complexType>
			<xs:complexContent>
				<xs:restriction base="xs:anyType">
					<xs:attribute name="myID" type="xs:ID" use="required"/>
					<xs:attribute name="base" use="required">
						<xs:simpleType>
							<xs:restriction base="xs:NMTOKEN">
								<xs:enumeration value="temperature"/>
								<xs:enumeration value="time"/>
								<xs:enumeration value="mass"/>
								<xs:enumeration value="amount"/>
								<xs:enumeration value="length"/>
							</xs:restriction>
						</xs:simpleType>
```

```
                                    </xs:attribute>
                                    <xs:attribute name="denominator" type="xs:string" use="required"/>
                                    <xs:attribute name="enumerator" type="xs:string" use="required"/>
                                </xs:restriction>
                            </xs:complexContent>
                    </xs:complexType>
            </xs:element>
            <xs:element name="PortType">
                    <xs:complexType>
                            <xs:choice minOccurs="0" maxOccurs="unbounded">
                                    <xs:element ref="Domain"/>
                                    <xs:element ref="VariableDefinition"/>
                            </xs:choice>
                            <xs:attribute name="name" type="xs:string" use="required"/>
                            <xs:attribute name="myID" type="xs:ID" use="required"/>
                    </xs:complexType>
            </xs:element>
            <xs:element name="ModelType">
                    <xs:complexType>
                            <xs:sequence>
                                    <xs:element ref="Annotation" minOccurs="0" maxOccurs="unbounded"/>
                                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                                            <xs:element ref="Domain"/>
                                            <xs:element ref="VariableDefinition"/>
                                    </xs:choice>
                                    <xs:element ref="SubmodelDefinition" minOccurs="0" maxOccurs="unbounded"/>
                                    <xs:element ref="PortDefinition" minOccurs="0" maxOccurs="unbounded"/>
                                    <xs:element ref="Coupling" minOccurs="0" maxOccurs="unbounded"/>
                                    <xs:element ref="Equation" minOccurs="0" maxOccurs="unbounded"/>
                            </xs:sequence>
                            <xs:attribute name="name" type="xs:string" use="required"/>
                            <xs:attribute name="myID" type="xs:ID" use="required"/>
                    </xs:complexType>
            </xs:element>
            <xs:element name="Domain">
                    <xs:complexType>
                            <xs:sequence>
                                    <xs:element ref="Expression"/>
                                    <xs:element ref="Expression"/>
                            </xs:sequence>
                            <xs:attribute name="myID" type="xs:ID" use="required"/>
                            <xs:attribute name="name" type="xs:string" use="required"/>
                            <xs:attribute name="symbol" type="xs:string" use="required"/>
                            <xs:attribute name="base.domain" type="xs:IDREF"/>
                            <xs:attribute name="type">
                                    <xs:simpleType>
                                            <xs:restriction base="xs:NMTOKEN">
                                                    <xs:enumeration value="DISCRETE"/>
                                                    <xs:enumeration value="CONTINUOUS"/>
                                            </xs:restriction>
                                    </xs:simpleType>
                            </xs:attribute>
                            <xs:attribute name="lower.bound.type">
                                    <xs:simpleType>
                                            <xs:restriction base="xs:NMTOKEN">
                                                    <xs:enumeration value="OPEN"/>
                                                    <xs:enumeration value="CLOSED"/>
                                            </xs:restriction>
                                    </xs:simpleType>
                            </xs:attribute>
                            <xs:attribute name="upper.bound.type">
                                    <xs:simpleType>
                                            <xs:restriction base="xs:NMTOKEN">
                                                    <xs:enumeration value="OPEN"/>
                                                    <xs:enumeration value="CLOSED"/>
                                            </xs:restriction>
                                    </xs:simpleType>
```

```
                </xs:attribute>
        </xs:complexType>
</xs:element>
<xs:element name="Distribution">
        <xs:complexType>
                <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                                <xs:attribute name="domain" type="xs:IDREF" use="required"/>
                        </xs:restriction>
                </xs:complexContent>
        </xs:complexType>
</xs:element>
<xs:element name="VariableDefinition">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Distribution" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
                <xs:attribute name="ref" type="xs:IDREF" use="required"/>
                <xs:attribute name="lowerBound" type="xs:string"/>
                <xs:attribute name="upperBound" type="xs:string"/>
                <xs:attribute name="unit" type="xs:string"/>
                <xs:attribute name="specification" default="STATE">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="CONSTANT"/>
                                        <xs:enumeration value="STATE"/>
                                        <xs:enumeration value="PARAMETER"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:complexType>
</xs:element>
<xs:element name="SubmodelDefinition">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Distribution" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
                <xs:attribute name="ref" type="xs:IDREF" use="required"/>
        </xs:complexType>
</xs:element>
<xs:element name="PortDefinition">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="StreamVariable" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
                <xs:attribute name="ref" type="xs:IDREF" use="required"/>
        </xs:complexType>
</xs:element>
<xs:element name="StreamVariable">
        <xs:complexType>
                <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                                <xs:attribute name="myID" type="xs:ID" use="required"/>
                                <xs:attribute name="model.variable" type="xs:IDREF" use="required"/>
                                <xs:attribute name="name" type="xs:string" use="required"/>
                        </xs:restriction>
                </xs:complexContent>
        </xs:complexType>
</xs:element>
<xs:element name="Coupling">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CoupledPort"/>
```

```
                        <xs:element ref="CoupledPort"/>
                    </xs:sequence>
                    <xs:attribute name="myID" type="xs:ID" use="required"/>
                    <xs:attribute name="name" type="xs:string" use="required"/>
                    <xs:attribute name="type" default="UNDIRECTED">
                        <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                <xs:enumeration value="UNDIRECTED"/>
                                <xs:enumeration value="DIRECTED"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="CoupledPort">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                            <xs:attribute name="port" type="xs:IDREF" use="required"/>
                            <xs:attribute name="submodel" type="xs:IDREF" use="required"/>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="Equation">
                <xs:complexType>
                    <xs:choice>
                        <xs:element ref="BalancedEquation"/>
                        <xs:element ref="ConditionalEquation"/>
                        <xs:element ref="StateTransitionNetwork"/>
                        <xs:element ref="LoopEquation"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
            <xs:element name="BalancedEquation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Distribution" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="Expression"/>
                        <xs:element ref="Expression"/>
                    </xs:sequence>
                    <xs:attribute name="myID" type="xs:ID" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="ConditionalEquation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="LogicalExpression"/>
                        <xs:element ref="Equation"/>
                        <xs:element ref="Equation"/>
                    </xs:sequence>
                    <xs:attribute name="myID" type="xs:ID" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="LoopEquation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Expression"/>
                        <xs:element ref="Expression"/>
                        <xs:element ref="Equation" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="myID" type="xs:ID" use="required"/>
                    <xs:attribute name="counter" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="StateTransitionNetwork">
                <xs:complexType>
```

```
                <xs:sequence>
                        <xs:element ref="Transition" maxOccurs="unbounded"/>
                        <xs:element ref="State" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
                <xs:attribute name="initialState" type="xs:IDREF" use="required"/>
        </xs:complexType>
</xs:element>
<xs:element name="State">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Equation" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
        </xs:complexType>
</xs:element>
<xs:element name="Transition">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="LogicalExpression"/>
                </xs:sequence>
                <xs:attribute name="myID" type="xs:ID" use="required"/>
                <xs:attribute name="fromState" type="xs:IDREF" use="required"/>
                <xs:attribute name="toState" type="xs:IDREF" use="required"/>
                <xs:attribute name="type" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="SYMMETRIC"/>
                                        <xs:enumeration value="ASYMMETRIC"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:complexType>
</xs:element>
<xs:element name="Expression">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Term" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="Term">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Factor" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="add.op" default="ADD">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="ADD"/>
                                        <xs:enumeration value="SUB"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:complexType>
</xs:element>
<xs:element name="Factor">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref="Number"/>
                        <xs:element ref="VariableOccurrence"/>
                        <xs:element ref="CounterOccurrence"/>
                        <xs:element ref="FunctionCall"/>
                        <xs:element ref="DomainOccurrence"/>
                        <xs:element ref="Expression"/>
                </xs:choice>
                <xs:attribute name="mul.op" default="NONE">
```

```
                        <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                <xs:enumeration value="MUL"/>
                                <xs:enumeration value="NONE"/>
                                <xs:enumeration value="DIV"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="Number">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                            <xs:attribute name="value" type="xs:string" use="required"/>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="ArrayIndex">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Expression"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="VariableOccurrence">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ComponentReference" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="ArrayIndex" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="definition" type="xs:IDREF" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="CounterOccurrence">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                            <xs:attribute name="loop.eqn" type="xs:IDREF" use="required"/>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="DomainOccurrence">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:restriction base="xs:anyType">
                            <xs:attribute name="domain" type="xs:IDREF" use="required"/>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="ComponentReference">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ArrayIndex" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="component" type="xs:IDREF" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="FunctionCall">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Expression" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="fcn.name" type="xs:string" use="required"/>
```

```
		</xs:complexType>
	</xs:element>
	<xs:element name="LogicalExpression">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="LogicalTerm" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="LogicalTerm">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="LogicalFactor" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="LogicalFactor">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Relation"/>
			</xs:sequence>
			<xs:attribute name="type" default="SIMPLE">
				<xs:simpleType>
					<xs:restriction base="xs:NMTOKEN">
						<xs:enumeration value="NEGATE"/>
						<xs:enumeration value="SIMPLE"/>
					</xs:restriction>
				</xs:simpleType>
			</xs:attribute>
		</xs:complexType>
	</xs:element>
	<xs:element name="Relation">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Expression"/>
				<xs:element ref="Expression"/>
			</xs:sequence>
			<xs:attribute name="type" default="NONE">
				<xs:simpleType>
					<xs:restriction base="xs:NMTOKEN">
						<xs:enumeration value="NONE"/>
						<xs:enumeration value="GREATER"/>
						<xs:enumeration value="EQUAL"/>
						<xs:enumeration value="GREATER.THAN"/>
						<xs:enumeration value="LESS.THAN"/>
						<xs:enumeration value="LESS"/>
					</xs:restriction>
				</xs:simpleType>
			</xs:attribute>
		</xs:complexType>
	</xs:element>
</xs:schema>
```