

# Uniform Comparison of Configuration Management Data Models

E. James Whitehead, Jr., Dorrit Gordon

Dept. of Computer Science  
University of California, Santa Cruz  
Santa Cruz, CA 95064 USA  
{ejw, dgordon}@cs.ucsc.edu

**Abstract.** The data models of a series of 11 configuration management systems—of varying type and complexity—are represented using containment data models. Containment data models are a specialized form of entity-relationship model in which entities may be containers or atoms, and the only permitted form of relationship is inclusion or referential containment. By using entities to represent the native abstractions of each system, and containment relationships to model inclusion and identifier references, systems can be modeled uniformly, permitting consistent cross-comparison of systems.

## 1 Introduction

The past 27 years of research and commercial development have witnessed the creation of scores of configuration management systems, with Conradi and Westfechtel’s survey listing 21 systems [3], and the CM Yellow Pages listing an additional 43 in the commercial sector [2]. Despite the existence of solid comprehensive survey work on configuration management systems [3], it is still the case today that developing a detailed understanding of the data model of a particular system requires labor-intensive study of the papers and documentation describing it. Worse, once developed, there is no modeling mechanism that allows this understanding to be represented in a way that both aids communication about the model, and permits cross-comparison with other system models. As a result, many of the data modeling lessons embedded in configuration management systems remain difficult to access for people outside the field.

Ideally, we would like to represent the data models of configuration management systems with a mechanism that has the following properties:

- *Uniformity*: model systems using a minimal and uniform set of abstractions. Instead of providing a normative definition of versioning and configuration management concepts, and then mapping system abstractions into these concepts, ideally we want to use atomic entities to build up models of each system’s abstractions.
- *Utility*: easily answer basic version data model questions such as, “how are version histories represented (if they are explicitly represented at all)”, “are

directories and other collection-type objects versioned”, and “is there some notion of workspace, activity, or configuration?”

- *Support Analysis*: be able to examine systems to tease out different design spaces employed in each system’s data model.
- *Graphic formalism*: communicate to a wide range of parties, inside and outside of the configuration management community, requiring minimal time to learn the graphic language, and with most people having an intuitive understanding of the formalism. Allow commonality among systems to be visually evident.
- *Concise format*: be able to fit the containment models of multiple systems onto a single page or screen, allowing rapid comparison of system data models.
- *Cross-discipline*: model the containment properties of multiple kinds of information systems, such as document management and hypertext versioning systems, and compare them to configuration management systems.

Previous work by the authors introduced the concept of containment data modeling, using it to describe a wide range of data models for hypertext, and hypertext versioning systems [21]. Other work began a preliminary examination of using containment modeling to represent configuration management systems, but modeled only a small number of systems [6]. Though this previous work suggested that containment modeling could successfully be applied to a wide range of configuration management systems, a larger set of systems must be modeled to provide full confidence that containment modeling is applicable in this domain. The present paper provides ten additional containment data models of configuration management systems, across a range of system types. Common features visible in the data models of each class of systems are then discussed.

In the remainder of this paper, we provide a brief introduction to containment modeling, and then apply it to the well known SCCS [12] and RCS [14] systems. Next, we model systems that provide an infrastructural data model, intended for use in creating more complex data models for specific uses or environments. Since these systems provide fundamental data model building blocks, they are a good test of the uniformity of containment data modeling. As a scalability test, the models of DSEE [9], ClearCase [8], and DeltaV [1] are presented, and aspects of the evolution of these complex models are discussed. The last set of models contrast the data models of two different system types, the version graph system CoMa [18], and the change-based system PIE [5], and demonstrate that containment models make system type differences much easier to identify and analyze. Related work and conclusions complete the paper.

## 2 Containment Modeling

Containment modeling is a specialized form of entity-relationship modeling wherein the model is composed of two primitives: entities and containment relationships. Each primitive has a number of properties associated with it. A complete containment model identifies a value for each property of each primitive. Unlike general entity-

relationship models, the type of relationships is not open-ended, but is instead restricted only to varying types of referential and inclusion containment.

## 2.1 Entity Properties

Entities represent significant abstractions in the data models of configuration management systems, such as source code revisions, directories, workspaces and configurations. The properties of entities are:

*Entity Type.* Entities may be either containers or atomic objects. Any entity that can contain other entities is a container. Any entity that cannot contain other entities is an atomic entity. Any entity contained by a container may be referred to as a containee of that container.

*Container Type.* There are two sub-types of container: 'and' and 'xor'. Unless otherwise specified, a container is generic. 'And' containers must contain two or more possible containee types and must contain at least one representative of each; 'xor' containers must have two or more possible containee types and may contain representatives of no more than one of them. While superficially similar to Tichy's notion of AND/OR graphs [13], the 'and' and 'xor' types herein describe constraints that apply to container classes, while AND/OR graphs are instance-level constraints. There is typically no precomputable mapping between the two sets of relationships.

*Total Containees.* A range describing the total number of containees that may belong to a container. A lower bound of zero indicates that this container may be empty.

*Total Containers.* Indicates the total number of containers to which this entity may belong. A lower bound of zero indicates that the entity is not required to belong to any of the containers defined in the model. Note that for the models in this paper, it is assumed that all elements reside in some type of file system, database, or other structure external to the system being modeled.

*Cycles.* Many systems allow containers to contain other containers of the same type. For example, in a standard UNIX file system, directories can contain other directories. This property indicates whether a container can contain itself (either directly or by a sequence of relationships). Unless otherwise specified this paper assumes that cycles are allowed, where possible.

*Constraints, Ordering.* The constraints property allows us to express other restrictions on the containment properties of an entity, such as mutual exclusion and mutual inclusion. The ordering property indicates whether there is an order between different containee types (i.e., class-level ordering; ordering of multiple containees of the same type is captured as a relationship property). In previous work [6] we used constraints and ordering in modeling characteristics of hypertext systems; in this paper we have not needed these properties.

## 2.2 Relationship Properties

*Containment Type.* Containment may be either inclusive or referential. Where inclusive containment exists, the containee is physically stored within the space

allocated for the container. This often occurs when an abstraction is used to represent actual content in the system. Referential containment indicates that the containee is stored independently of the container.

*Reference Location.* The references that connect containers and their containees are usually stored on the container; but on some occasions they are stored on the containee, or even on both container and containee. This property indicates where the references are stored.

*Membership.* The membership property indicates how many instances of a particular container type an entity may belong to. If the lower bound on the value is zero, then the entity is permitted, but not required, to belong to the container indicated by the relationship.

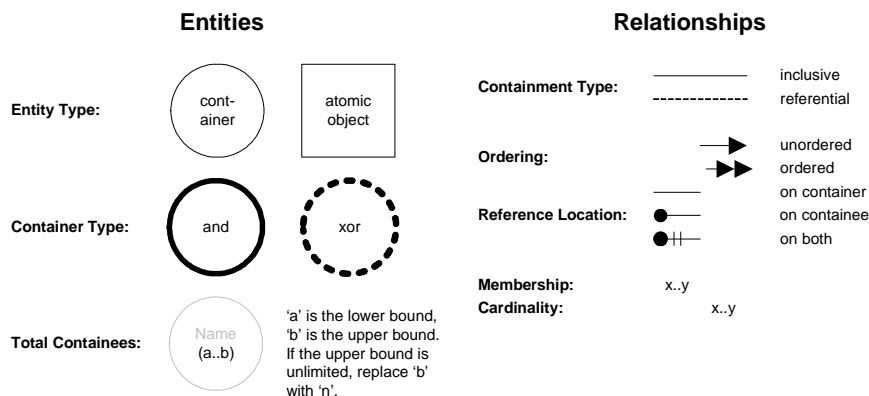
*Cardinality.* The cardinality property indicates how many instances of a particular containee a container may contain. If the lower bound on the value is zero, then the container is permitted, but not required, to contain instances of the containee indicated by the relationship.

*Ordering.* The ordering property indicates whether there is an ordering between instances of a particular entity type within the container indicated by the relationship.

### 2.3 Graphical Notation

Entities are represented as nodes of the containment model graph, and relationships are the arcs. Figure 1 below describes the graphic notation used to represent containment data models. Relationships are indicated by directed edges between pairs of entity nodes in the containment model graph. In Figure 1, visual elements that represent the source end of an edge are shown on the left ends of lines. Visual elements that represent the sink end of an edge appear at the right ends of lines.

To readers familiar with the Unified Modeling Language (UML), a natural question is why we did not use this better-known modeling notation instead. There are several reasons. First, by definition all UML classes may potentially have attributes.



**Figure 1** – Graphical notation used to represent containment data models.

In our modeling of hypertext systems in [21] we found several systems with content objects that, by definition, could not have associated attributes. UML makes no visual distinction between container and non-container objects, and since containment is central to our approach, we wished to make this more explicit by representing containers as circles, and atoms as squares.

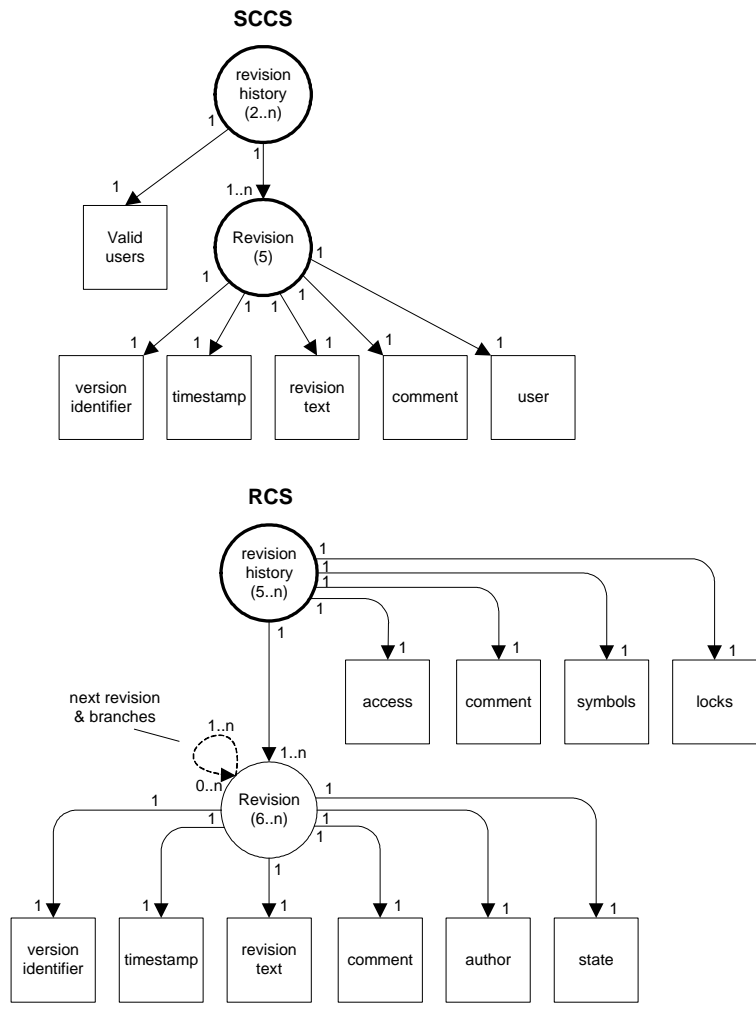
UML contains many features that are not needed for containment modeling, such as the visibility of attributes, and operations available on data. While the core of UML is simple to understand, it certainly takes more time to learn than our notation. UML also permits the expression of inheritance relationships, which we do not permit. In previous work we used inheritance in a limited way within containment diagrams [21]. In later work, we moved away from this, since we found it difficult to comprehend that all relationships that apply to the parent also apply to the child [6]. Our experience has been that not using inheritance leads to minor duplication of entities and containment relationships within diagrams, and these diagrams are much easier (for us) to understand.

It is possible to represent containment data models using UML notation, and in previous work we have found that the equivalent UML model takes up more paper space than the equivalent containment diagram [6]. A system designer familiar with UML could use UML for containment modeling with minimal loss of fidelity, and the advantages of compatibility with UML might outweigh the advantages of the notation presented herein.

### **3 Modeling Version Control Systems**

In previous work, we have created containment models of hypertext systems [21], as well as a small set of hypertext versioning and configuration management systems [6]. Due to the complexity and variety of existing configuration management systems, we want to model a larger number of these systems, both to validate that containment modeling is sufficiently expressive to represent their data models, as well as to support an initial cross-comparison of data modeling approaches. To begin with, we present containment data models of RCS [14] and SCCS [12], two well-known version control systems, in Figure 2 below.

Examining Figure 2 highlights several aspects of containment modeling. First, notice that individual revisions are modeled as containers, rather than atomic entities. Since revisions have associated with them a number of attributes, as well as the textual content of the revision, this is modeled using inclusion containment relationships. That is, a revision is viewed as a container of a number of predefined attributes and the revision text. The relationship between the revision history and individual revisions is also modeled using inclusion containment, since it is not possible to delete the revision history without also deleting all revisions. Each revision also has one or more identifiers that point to the next revision, and emanating branches, and this use of identifiers to point at other entities is the characteristic quality of referential containment. Hence, even though RCS inclusively contains revisions within a revision history, there is still referential containment within its data model.



**Figure 2** – Data model of the SCCS [12] and RCS [14] systems.

Finally, neither model explicitly represents deltas. This is consistent with the user interface of both tools, which allow users to operate on revisions but not deltas, and agrees with Conradi and Westfechtel’s characterization of these systems as state-based [3]. That revision histories and revisions are stored in files, and revisions are stored as deltas, can be viewed as *concrete representation* issues. While these concrete representation issues have an important impact on the performance of the tools (e.g., reverse deltas vs. forward deltas [14]), they are a concern that can be abstracted away when examining data models.

With implementation representation issues abstracted away, it is easy to identify differences and similarities between these two systems. The most significant difference is the referential containment arc in RCS, which points to branches and the

next revision. This permits RCS to have a much richer set of branch operations than SCCS. The other significant difference lies in the attributes of revision histories and revisions. In RCS, the revision history contains a set of symbols that are used to label specific revisions; SCCS lacks this capability. Only RCS revisions contain a state attribute, thus permitting RCS to associate revisions with specific states in a (externally defined) state-transition workflow. RCS stores information about locks as an attribute of the version history, while SCCS does not save any lock information.

Both systems share many similarities. Each has a revision history inclusively containing revisions, and the revisions in both systems share many of the same attributes (identifier, timestamp, revision text, comment, author). In both systems, access control information is stored on the revision history (SCCS: valid users, RCS: access).

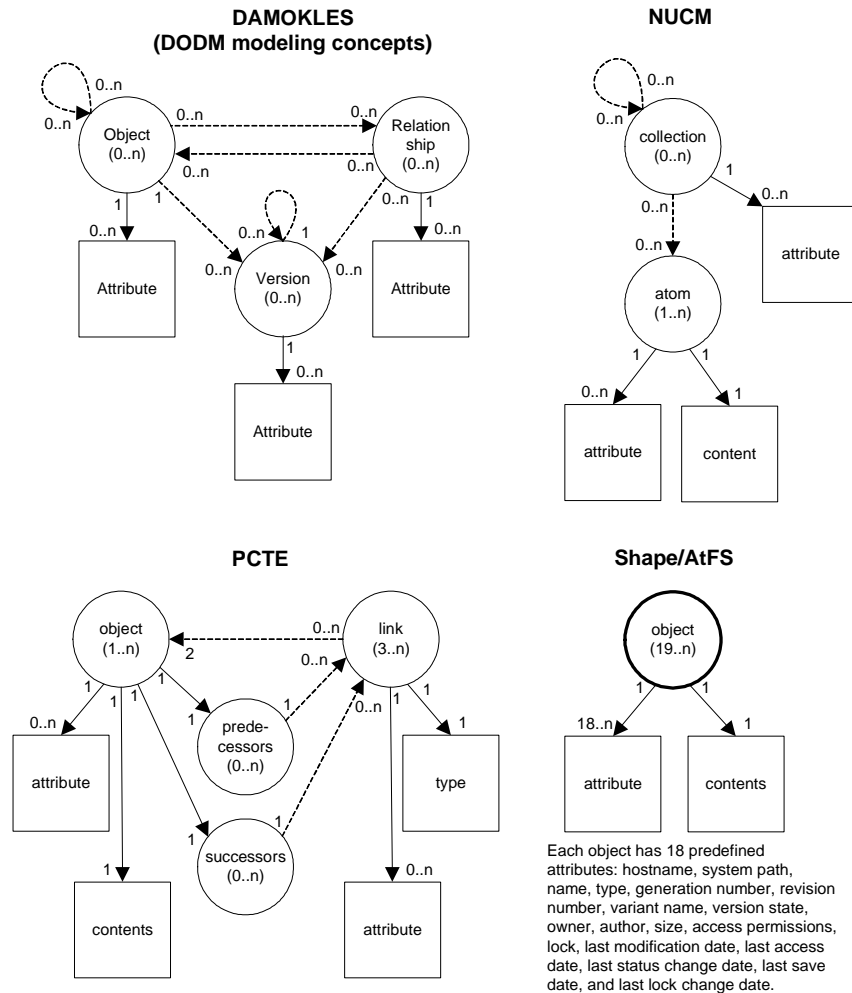
Equally important, by examining Figure 2 one can clearly see what features these systems lack. Neither has an entity representing workspaces, a first-class notion of configuration, logical change tracking (e.g., activities or change sets), or an ability to record the revision history of directories. Additionally, both systems only allow predefined attributes, and do not allow arbitrarily named attributes. While these facts are well known, what is unusual is how quickly they can be determined by examining the containment data model. Instead of spending many minutes scanning through the documentation and system papers, the containment data model allows these questions to be answered quickly. Similarly, while it is possible to provide a detailed comparison of these data models by laboriously studying the documentation of the SCCS and RCS file formats, it is much faster to just contrast two containment diagrams. Additionally, when many systems are to be compared, it is extremely difficult to remember every detail of each system's data model. A consistent, common notation is crucial.

## **4 Models of Configuration Management Systems**

In order to demonstrate that it is possible to create containment models of a wide range of configuration management systems of varying complexity, we now present a series of such models.

### **4.1 Data Models of Infrastructure Systems**

A major goal of containment modeling is uniformity, that is, representing data models while bringing as few biases into the modeling activity as possible. By using a simple model with limited prejudices, it permits expression of a wide range of data models using a uniform set of modeling primitives. One way to validate this quality of uniformity is to model existing systems whose goal is to provide a simple, infrastructural data model used as a basic building block for creating more complex data models for a specific application or programming environment. If it is possible to create uniform containment models of infrastructural data models, then cross comparison of these models will be possible, permitting an examination of the



**Figure 3** – Data models of DAMOKLES [4], NUCM [16], PCTE [11,17], and Shape/AtFS [10].

modeling tradeoffs made by each. Examples of such systems are DAMOKLES [4], PCTE [11,17], NUCM [16], and Shape/AtFS [10], modeled in Figure 3.

What immediately jumps out from the figure is the relative simplicity of the data models of these systems, ranging from 1 to 4 container entities. This is due to their explicit desire to have small set of entities that can then be used to build up more complex capabilities. All systems have the ability to represent version histories, however each system does so differently. DAMOKLES and NUCM can both use containment to represent version histories, though DAMOKLES has a compound object containing a series of versions (and versions containing versions), while NUCM makes it possible to use an explicit collection entity. In PCTE, every object

maintains lists of links that point to predecessor and successor objects, and NUCM collections could also be used to represent predecessor/successor relationships. Neither DAMOKLES nor Shape/AtFS explicitly represent predecessor/successor relationships, with Shape/AtFS representing version histories implicitly via version identifier conventions (e.g., version 1.2 defined to be the successor of 1.1).

While the systems in Figure 3 are generally capable of representing the data models of the others, they do so with varying degrees of mismatch. For example, DAMOKLES objects and relationships map to NUCM collections, and DAMOKLES versions map somewhat to NUCM atoms. However, DAMOKLES requires versions to belong to only one object; NUCM doesn't have this restriction. DAMOKLES objects and versions store all non-containment state in attributes, while NUCM atoms have a distinguished content chunk. Since PCTE uses links to represent containment relationships, and NUCM uses collections to represent links, it is possible to map between NUCM and PCTE by performing appropriate link/collection conversions. While Shape/AtFS could mimic collections and relationships by defining attributes on Shape/AtFS objects, these would not have any special properties, such as the referential integrity constraints enforced by PCTE on containment links.

As with RCS and SCCS, the containment data models in Figure 3 permit a rapid examination of facilities not provided by these systems. None of the systems provide explicit support for workspaces, activities/change sets, or configurations, though these facilities could be provided by building on the existing data models.

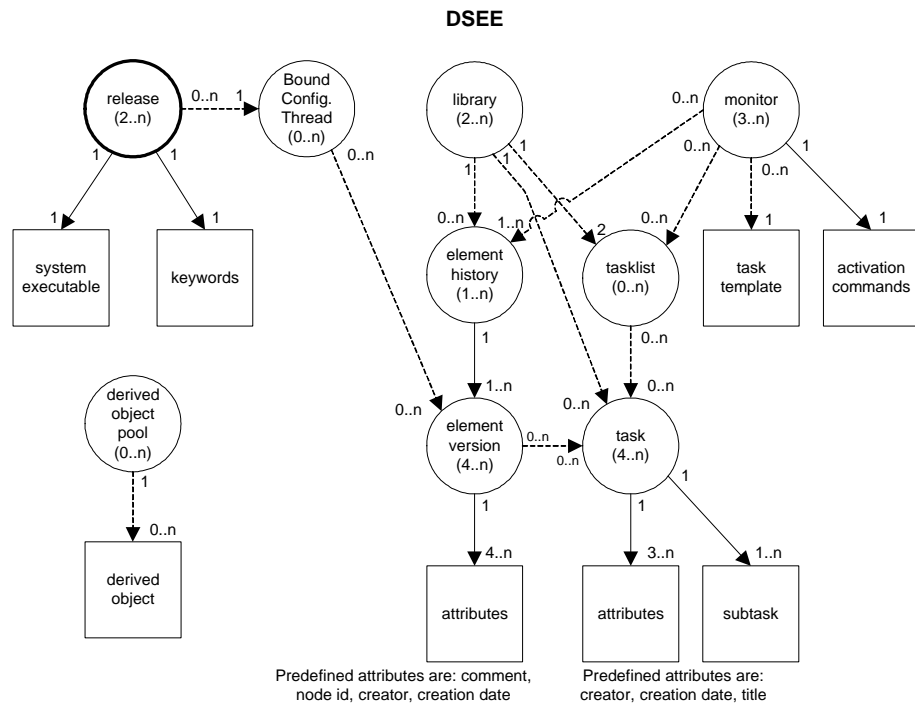
Since all models can be represented and compared using containment modeling, it highlights the ability of containment modeling to allow cross comparison of even basic, foundational models, permitting the assumptions of each to be compared.

## 4.2 Tracing the Evolution of Data Models

Since all of the systems modeled so far have been relatively simple, it is reasonable to be curious about how well containment modeling handles more complex systems. We address this issue by examining the DSEE [9], ClearCase [8,19], and DeltaV [1] systems, modeled in Figure 4-Figure 6. Together they form a direct intellectual lineage, with DSEE and ClearCase having the same initial system architect, David Leblang, and with ClearCase influencing DeltaV via the direct involvement of Geoff Clemm, who was simultaneously an architect of both ClearCase and DeltaV from 1999-2002.

Since these systems all have complex data models, ranging from 9 to 14 collection entities, the models validate that containment modeling can represent large systems. By concisely representing the data models of these systems in three pages, we can view the evolution of system concepts in this lineage more easily than by reading over 150 pages of primary source text.

All three systems have objects that represent an entire revision history, and though all systems supporting branching, only ClearCase has an entity that explicitly represents branches. While DSEE and ClearCase both use delta compression, in the DSEE data model element revisions are inclusively contained by the element history, while ClearCase uses referential containment between elements, branches, and versions.



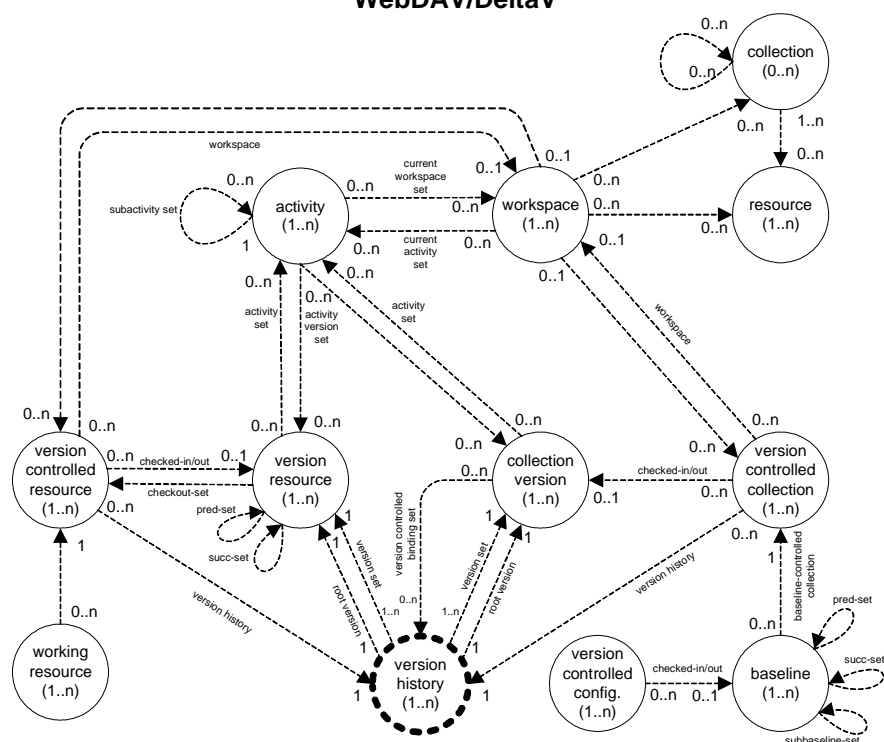
**Figure 4 - Data model of the DSEE system [9].**

The DSEE notion of task and tasklist evolved into the ClearCase and DeltaV notion of activity, though in DSEE a revision points at an activity, while in ClearCase and DeltaV the activity points at revisions. In ClearCase and DeltaV, activities are more integrated into version selection and workspaces than in DSEE, with ClearCase streams pointing to both activities and views, and DeltaV activities pointing directly to workspaces (and vice-versa). The dynamic version selection provided by DSEE bound configuration threads (BCTs) and ClearCase streams, views, and configuration specifications are similar, though ClearCase gains additional flexibility for handling large projects by introducing the stream entity. DeltaV supports only version selection using baselines and activities, with no equivalent concept to BCTs or views.

Both DSEE and ClearCase provide abstractions for representing groupings of multiple source objects and directories. DSEE has the notion of a library, containing a set of elements and tasklists, appropriate for modeling subsystems of a large project. ClearCase offers greater flexibility, with a distinction between Project Versioned Object Base (PVOB) and regular VOBs. PVOBs hold project and component entities, while VOBs hold source objects and their inter-relationships. A large project might have multiple VOBs, and a single PVOB, thus proving scalability advantages over DSEE. The implicit notion of a “server” in DeltaV is similar to a combined PVOB/VOB.



## WebDAV/DeltaV



**Figure 6** – Data model exposed by the DeltaV protocol [1]. To reduce figure clutter, resource properties (attributes) are not shown. For many DeltaV entities their properties contain lists of identifiers, and hence strict modeling would show these properties as container entities. Due to the complexity of the model, we use a convention of annotating containment arcs with the name of the property that contains the identifiers for that arc. Plain arcs represent containment by the entity proper (i.e., not by a property).

models grow larger, our choice to label entities with the same terminology used in the description of the system makes it more difficult to understand each diagram. While using each system's native terminology avoids modeling bias and saves us from a thorny definitional thicket, it also requires the reader to have greater knowledge of each system to understand the diagram. This was not as great a problem for the systems in Figure 3, since their data models are less complex. In future work, we hope to distill out design spaces for such issues as representing workspaces and configurations, thereby focusing on smaller, more manageable aspects of complex systems.

### 4.3 Comparing Different Types of Configuration Management System

In our last set of figures, we present the data models of CoMa [18] and PIE [5] in Figure 7. Using the taxonomy of systems provided in [3], CoMa is a version graph system, while PIE is a change-based versioning system. Since these are two different styles of system, we would expect these differences to be visible in a comparison of their containment models, and this is indeed the case.

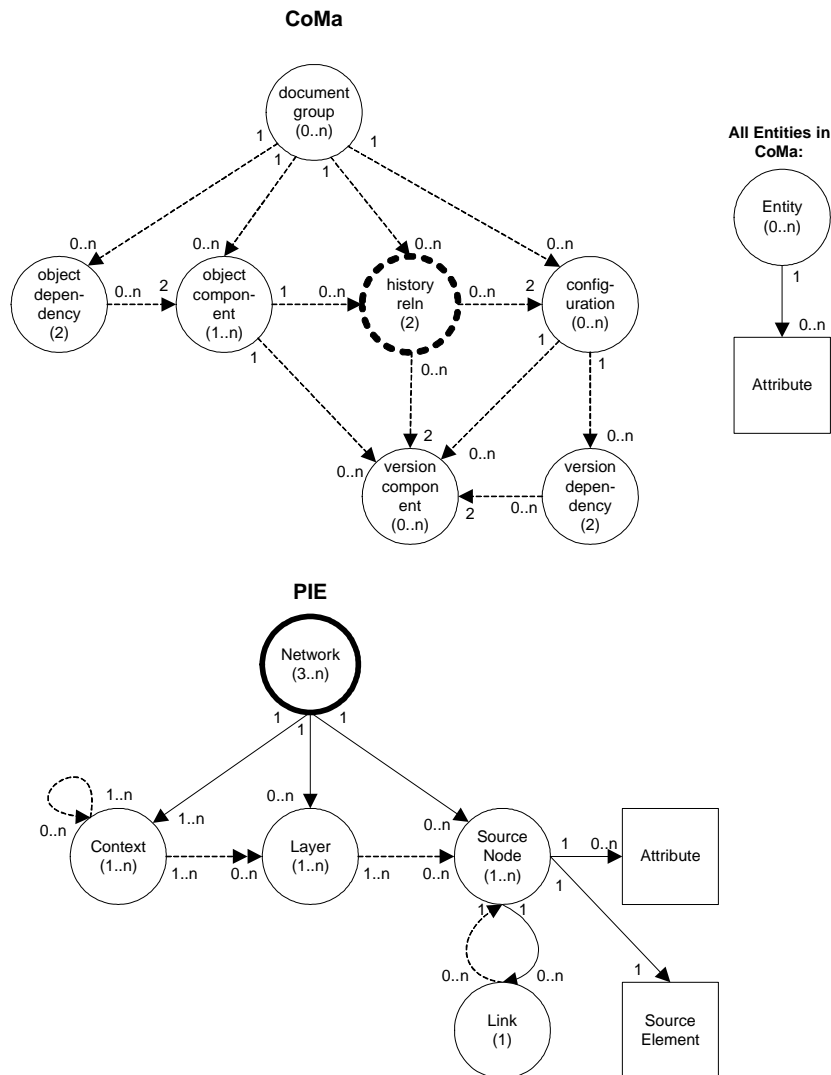


Figure 7 – Data models of the CoMa [18] and PIE [5] systems.

The CoMa system represents a version history using a versioned object container that holds all revisions, as well as links for the predecessor/successor relationships. In Figure 7 this pattern is visible twice, first as the object component (the versioned object) containing version components (individual revisions of objects) and history relationships (representing predecessor/successor relationships), next as document groups (versioned object) containing configurations (revisions) and history relationships. PIE uses a different scheme altogether, with individual source nodes organized into layers representing versions or variants, and contexts grouping layers, with occlusion among layers depending on their relative ordering.

## 5 Related Work

Several prior research efforts are similar to the work presented herein. Conradi and Westfechtel's survey of configuration management systems examines many issues of data modeling across a broad set of systems [3]. Our work differs in that it focuses on data modeling, and provides a modeling mechanism that permits uniform cross-comparison of data models. Conradi and Westfechtel's description of data modeling issues was primarily textual, as contrasted with the graphical technique presented here. However, their work also surveyed version selection techniques, which are not covered by containment modeling.

The schema languages in CoMa [18] and DAMOKLES [4] bear some similarity to the containment modeling approach, having notions of containment, and a separation between the schema and its concrete representation. However, neither of these schemas have been used to model existing CM systems. The data model of NUCM [15,16] is also similar to the basic set of primitives in containment modeling, since it too has containers and atoms. Since the intended use of NUCM is to construct more sophisticated configuration management capability on top of these basic entities, there is no separation between model and concrete representation, with NUCM directly representing containers and atoms in its repository. Containment modeling is intended to be more abstract, and hence doesn't have a default concrete representation. Another difference is that NUCM collections and atoms have hardwired attributes, while there are no predetermined inclusion containment relationships among entities in a containment data model. However, given that the vast preponderance of collection entities in configuration management systems have attributes, it is arguable whether the ability to explicitly model the inclusion of attributes is a significant benefit. We have primarily found this flexibility to be useful in modeling hypertext systems, where the use of attributes is not quite so universal.

## 6 Contributions

Containment modeling is a modeling mechanism that permits uniform representation of the data models of a wide range of configuration management systems. Containment modeling has been validated by presenting the models of 11 existing configuration management systems. In the case of ClearCase and DeltaV, these are

very complicated data models, perhaps the most complex in existence. Additionally, in previous work we have created containment models of an additional 16 hypertext systems [6,21], highlighting how the uniformity of the modeling technique permits data modeling across multiple system domains.

Containment modeling provides a new technique that is useful for performing detailed comparisons of configuration management data models. Since containment data models are visual and compact, it is much easier to cross-examine data models, making it possible to quickly identify similarities and differences. In previous work we have used insights derived from containment models of hypertext versioning systems to develop design spaces for link and structure versioning [20], and in future work we wish to similarly leverage configuration management containment models to develop design spaces for features such as workspaces, change sets, and configurations.

The compact, visual notation of containment models is good for capturing the current best understanding of a system's data model. This permits iterative improvement of containment data models over time, as understanding improves. This is not possible without an explicit model. Additionally, the process of creating models is error-prone, indicative of the fact that trying to understand a data model from a textual description is also error-prone. While we have worked to remove errors from the containment models presented herein, there is no mechanical way to validate their correctness, and hence it is possible some errors may still persist. The explicit nature of containment models allows errors to be exposed, and then corrected.

Lastly, the visual notation makes it easier to communicate results across disciplinary boundaries. The existence of 27 data models of configuration management and hypertext systems makes it easier to compare/contrast across these two domains, and learn from each other. In future work we hope to extend the containment data modeling technique into other domains, including Document Management, and VLSI CAD [7], as well as to model more CM systems, to complete our data model survey of CM systems.

## **Acknowledgements**

This research was supported by the National Science Foundation, by grant NSF CAREER CCR-0133991.

## **References**

- [1] G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead, "Versioning Extensions to WebDAV," Rational, IBM, Microsoft, U.C. Santa Cruz. Internet Proposed Standard Request for Comments (RFC) 3253, March, 2002.
- [2] CM Today, "CM Yellow Pages," (2002). Accessed December 21, 2002. [http://www.cmtoday.com/yp/configuration\\_management.html](http://www.cmtoday.com/yp/configuration_management.html).
- [3] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, no. 2 (1998), pp. 232-282.

- [4] K. R. Dittrich, W. Gotthard, and P. C. Lockemann, "DAMOKLES - A Database System for Software Engineering Environments," *Proc. Advanced Programming Environments*, Trondheim, Norway, June, 1986, pp. 353-371.
- [5] I. P. Goldstein and D. P. Bobrow, "A Layered Approach to Software Design," in *Interactive Programming Environments*, New York, NY: McGraw-Hill, 1984, pp. 387-413.
- [6] D. Gordon and E. J. Whitehead, Jr., "Containment Modeling of Content Management Systems," *Proc. Metainformatics Symposium 2002 (MIS'02)*, Esbjerg, Denmark, Aug 7-10, 2002.
- [7] R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases," *Computing Surveys*, vol. 22, no. 4 (1990), pp. 375-408.
- [8] D. Leblang, "The CM Challenge: Configuration Management that Works," in *Configuration Management*, New York: Wiley, 1994, pp. 1-38.
- [9] D. B. Leblang and J. R. P. Chase, "Computer-Aided Software Engineering in a Distributed Workstation Environment," *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Pittsburgh, PA, April, 1984, pp. 104-112.
- [10] A. Mahler and A. Lampen, "An Integrated Toolset for Engineering Software Configurations," *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symp. on Practical Software Development Environments*, Boston, MA, Nov. 28-30, 1988, pp. 191-200.
- [11] F. Oquendo, K. Berrada, F. Gallo, R. Minot, and I. Thomas, "Version Management in the PACT Integrated Software Engineering Environment," *Proc. ESEC'89*, Coventry, UK, Sept. 11-15, 1989, pp. 222-242.
- [12] M. J. Rochkind, "The Source Code Control System," *IEEE Transactions on Software Engineering*, vol. 1, no. 4 (1975), pp. 364-370.
- [13] W. F. Tichy, "A Data Model for Programming Support Environments and its Application," *Proc. IFIP WG 8.1 Working Conf. on Automated Tools for Info. Systems Design and Dev.*, New Orleans, LA, Jan 26-28, 1982, pp. 31-48.
- [14] W. F. Tichy, "RCS - A System for Version Control," *Software-Practice and Experience*, vol. 15, no. 7 (1985), pp. 637-654.
- [15] A. van der Hoek, "A Generic Peer-to-Peer Repository for Distributed Configuration Management," *Proc. ICSE-18*, Berlin, 1996, pp. 308-317.
- [16] A. van der Hoek, "A Testbed for Configuration Management Policy Programming," *IEEE Trans. Software Eng.*, vol. 28, no. 1 (2002), pp. 79-99.
- [17] L. Wakeman and J. Jowett, *PCTE: The Standard for Open Repositories*. New York: Prentice Hall, 1993.
- [18] B. Westfechtel, "Using Programmed Graph Rewriting for the Formal Specification of a Configuration Management System," *Proc. 20th Int'l Workshop on Graph-Theoretic Concepts in Computer Science (WG'94)*, Herrsching, Germany, June 16-18, 1994, pp. 164-179.
- [19] B. A. White, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*. Boston, MA: Addison-Wesley, 2000.
- [20] E. J. Whitehead, Jr., "Design Spaces for Link and Structure Versioning," *Proc. Hypertext 2001*, Århus, Denmark, August 14-18, 2001, pp. 195-205.
- [21] E. J. Whitehead, Jr., "Uniform Comparison of Data Models Using Containment Modeling," *Proc. Hypertext 2002*, College Park, MD, June 11-15, 2002.