Containment Modeling of Content Management Systems

Dorrit H. Gordon and E. James Whitehead Jr.

Department of Computer Science
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, California 95064
{dgordon, ejw}@soe.ucsc.edu

Abstract. Containment models are a specialized form of entity-relationship model in which the only allowed form of relationship is a 'contains' relationship. This paper builds on the authors' previous work on containment modeling by clarifying the graphical notation and increasing the expressiveness of the model in a way that expands the technique to systems outside the hypertext domain. Examples from the hypertext, hypertext versioning, and configuration management domains illustrate the cross-domain applicability of this technique.

1 Introduction

The past 35 years have witnessed the emergence of a broad class of systems to manage digital content. Under the banners of Hypertext, Software Configuration Management (SCM), Document Management, VLSI Computer Aided Design (CAD), and Media Asset Management, theses systems have been developed to address the distinctive requirements of each community. Despite the many differences among the systems in these domains, a current of commonality runs through them all: each of these systems provide a repository that contains all content, and within this repository content can have associated metadata, and can be aggregated together into collections. These systems commonly offer change control functionality: facilities for versioning content, collections, and sets of these. The systems provide multi-person write access and hence have capabilities for group work, ranging from per-object concurrency control to multi-object collaborative workspaces.

To date, it has been difficult to explore the commonality among these disparate systems since there is no easy way to compare their data models. In our previous work, we introduced a novel modeling technique that focuses on expressing the containment relationships between entities in hypertext systems [11, 12]. This *containment modeling* technique has been used to describe the data models of a broad range of hypertext systems, including monolithic (open) hyperbase, linkbase, and hypertext versioning [9], as well as the Dexter reference model [6] and the Web with WebDAV extensions [13]. When applied to hypertext systems, containment modeling provides several benefits, including the ability to concisely represent complex data models, and easily cross-compare the data models of multiple systems. It provided a data modeling language that could be used to describe and reason about sets of hypertext systems, rather than single systems.

In this paper, we expand the descriptive scope of containment data modeling to encompass both Software Configuration Management systems, and Aquanet, the first spatial hypertext system. Aquanet represents a class of hypertext systems not previously modeled. In the process of modeling these systems, we make multiple refinements to containment data modeling to adequately capture the nuances of each system. Together, these refinements form the primary contribution of this work: extending the scope of containment modeling to handle systems outside the hypertext domain. In so doing, we demonstrate that containment data modeling is cross-domain, capable of modeling content management systems in general, not just hypertext systems. Furthermore, by modeling multiple SCM, hypertext, and hypertext versioning systems in one place, in a consistent notation, we are able to perform a cross-domain comparison of the data models of these systems.

In the remainder of the paper, we provide a description of the enhanced containment data modeling technique, including the semantics of containment, and a graphical notation for their visual representation. We validate containment data modeling by using it to model five hypertext systems representative of the monolithic, spatial, and hyperbase systems, along with three SCM systems, and one hypertext versioning system. Together, these models highlight the containment model's ability to represent complex data models of a range of content management systems. After each set of models, the paper discusses observed similarities and differences among the systems. We conclude with a brief look at future directions for this research.

2 Containment Modeling

Containment modeling is a specialized form of entity-relationship modeling wherein the model is composed of two primitives: entities and 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 inclusive containment.

2.1 Entity Properties

Entities represent significant abstractions in the data models of content management systems. For example, when modeling hypertext systems, entities represent abstractions such as works, anchors, and links, while in configuration management systems, they represent abstractions such as versioned objects and workspaces. The properties of entities are:

Entity Type. Entities may be either containers or atomic objects. Any entity which can contain other entities is a container. Any entity which cannot container other entities is an atomic object. Any entity which is contained in 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 (i.e. neither 'and' nor 'xor'). '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.

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

Total Containers. A range describing 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 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 filesystem, 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 existing containment relationships make them possible).

Ordering. The ordering property indicates whether there is an order between different containee types (ordering of multiple containees of the same type is captured as a relationship property). For example, a full name contains a first name, zero or more middle names, and a last name. The ordering property on the full name container indicates that the first name goes before the middle names which go before the last name. The importance of the ordering of the middle names with respect to one another is captured by the ordering property on the relationship connecting full name to middle name.

Constraints. The constraints property allows us to express other restrictions on the containment properties of an entity. Two kinds of constraints currently identified are mutual exclusion and mutual inclusion.

A mutual exclusion constraint on a container indicates that it may contain one or the other, but not both, of a pair of possible containees. This occurs in configuration management systems where a special entity (the versioned object) exists to hold versions of an object. In this case, an instance of a versioned object holds versions of only one object type, although there may be several objects which are versioned in this way.

A mutual inclusion constraint on a container indicates that it must contain both (or neither) of a pair of possible containees. For example, in a system which maintains metadata about its contents, it might be possible to create an empty repository, but it would be impossible to create an item in the repository without creating metadata and it would be impossible to create metadata without creating an item.

These constraints have been described as they apply to containers. They may also apply to entities with respect to their containers. That is, there may be a mutual exclusion constraint on an entity which indicates that if it belongs to one container, it may not belong to another.

The mutual inclusion constraint can also be broken down so that it is unidirectional. For example, a graph can contain nodes without containing edges, but it cannot contain edges without containing nodes.

2.2 Relationship Properties

Containment Type. Containment may be either inclusive or referential. Where inclusive containment exists, the containee is physically stored within the spaced allocated for the container. This occurs, for example, when an abstraction is used to represent actual content (as opposed to structure) in the system. Referential containment indicates that the containee is stored independently of the container.

Reference Location. The references which 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. FOR example, full names contain zero or more middle names. Middle names are ordered in the full name container. Mary Jane Elizabeth Smith is not equivalent to Mary Elizabeth Jane Smith. We could imagine entering everyone's middle names into a drawing for a prize. The collection of entries (assuming no one's cheating) would be unordered.

Containee Type. This property is unique to versioning systems. It distinguishes between relationships where the container contains multiple versions of a single containee, single versions of multiple containees, or multiple versions of multiple containees. This only becomes important when the cardinality of the relationship is greater than one. It is important because many configuration management systems have examples of two or more of these cases.

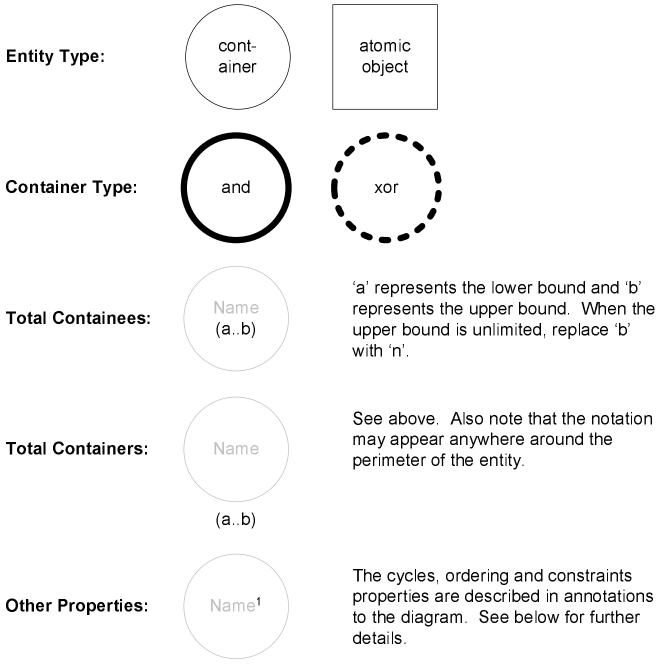
As an example of the first case, some configuration management systems use an abstraction of an object to contain all the version of that object. There might be many objects in the system, but each abstraction contains version of only one object.

As an example of the second case, configuration management systems all have some mechanism for selecting the specific version of a number of objects to include in a release.

As an example of the third case, some configuration management systems allow users to construct spaces with any contents they desire (potentially including multiple objects and multiple versions of those objects).

2.3 Graphical Notation

Entities and Their Properties Entities are represented as nodes of the containment model graph. The following legend shows the representation of entity properties on the graph.



'a' represents the lower bound and 'b' represents the upper bound. When the upper bound is unlimited, replace 'b' with 'n'.

See above. Also note that the notation may appear anywhere around the perimeter of the entity.

The cycles, ordering and constraints properties are described in annotations to the diagram. See below for further details.

Fig. 1. Entity Legend

Unless otherwise specified, it is assumed that, where possible, cycles are permitted. If cycles are not permitted, the statement *no_cycles* appears in a container's annotations.

If there is an ordering among different containee types in a container, the ordering is described by an ordering statement. Ordering statements have the following syntax: *order*(containee 1, containee 2, ..., containee n). The containees are listed in order. Only those containees involved in a partial ordering are listed. If there are multiple partial orderings, the annotation contains one ordering statement per partial ordering.

Constraints are defined using predicate logic. The most common predicate is *contains*(container, containee). A mutual exclusion constraint would appear as follows: *contains*(container, containee 1) ↔ ¬*contains*(container, containee 2).

Relationships and Their Properties Relationships are indicated by directed edges between pairs of entity nodes in the containment model graph. In the following legend, 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.

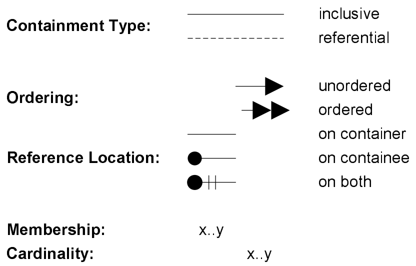


Fig. 2. Relationship Legend

In models of versioning systems the containee type is just in front of the cardinality, followed by a colon. A 'V' is used for version, an 'O' for object, and a 'VO' for both. For example, if the container may contain one or more versions of a containee, the containee type and cardinality would be noted on the diagram as, "V:1..n."

Why Not UML? It is possible to draw a containment model using the standard Unified Modeling Language (UML) notation, so why have we chosen to develop a different notation? There are two reasons: readability and utility. Figure 3 shows two different versions of the same simple containment model: one drawn using UML, the other drawn with the notation presented in this paper. The UML diagram has considerably more elements than the diagram that uses our notation, because relationship properties beyond cardinality and membership must be described using UML's annotation mechanism. Applied to a more complex model, UML notation produces a cluttered, overly complex diagram that is difficult to read, and which may be too large to print or display conveniently.

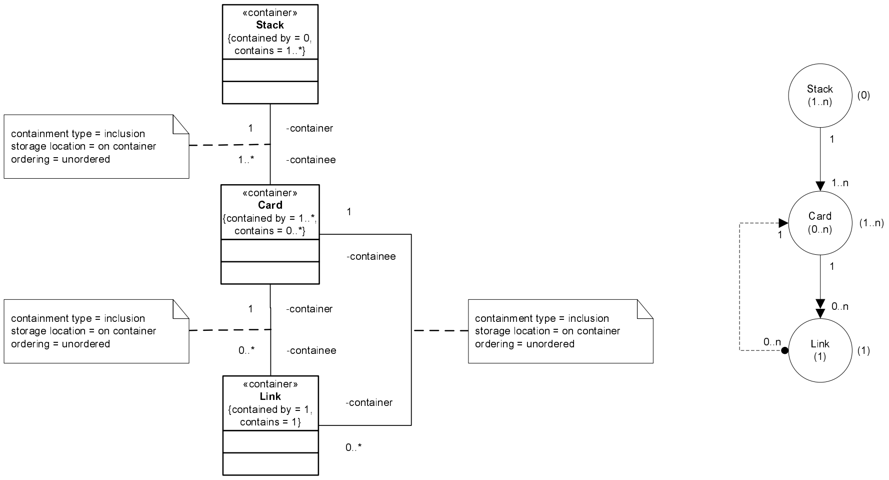

«container»
Stack
{contained by = 0, contains = 1..*}
containment type = inclusion
storage location = on container
ordering = unordered
1
-container
1..*
-containee
«container»
Card
{contained by = 1..*, contains = 0..*}
1
-containee
containment type = inclusion
storage location = on container
ordering = unordered
1
-container
0..*
-containee
containment type = inclusion
storage location = on container
ordering = unordered
«container»
Link
{contained by = 1, contains = 1}
-container
0..*
Stack
(1..n)
(0)
1
1..n
Card
(0..n)
(1..n)
1
1
0..n
0..n
Link
(1)
(1)

Fig. 3. UML and non-UML representation of the HyperCard [1] containment model.

The UML representation is also less useful than a representation using the notation we have developed. By describing as many properties as possible graphically, we enable the user to quickly and easily compare models of different systems. The visual representation allows users to quickly and easily spot the similarities and differences between systems. In the UML notation, a user would need to laboriously compare the annotations on each relationship to determine whether patterns existed.

Although we do not believe the standard UML notation is sufficient for containment modeling, it would be possible to frame the notation described here as a UML extension. The stereotyping mechanism in UML allows the creation of new types of objects and allows the assignment of unique visual representations for those types.

2.4 Changes to Containment Modeling

In this paper, we have improved upon the containment modeling technique previously described in [12], in two ways. First, the graphical representation now shows each axis of variability in the model as a single axis of variability in the diagram. For example, ordering is shown by having either one arrowhead (unordered) or two (ordered); previously, a circle at the tail of a containment relationship showed ordering. This frees the circle to be used exclusively to represent the location of the reference identifier in the referential containment.

Second, we chose to eliminate inheritance and storage relationships from the modeling system in [12]. Inheritance was only used occasionally, and then only to reduce clutter on diagrams. When appropriate, it can be represented by a separate inheritance hierarchy, such as a UML structure diagram. Storage relationships were mainly used to relate the system to the external structure (such as a file system) where it resided. A gain, this may be valuable information, but is not truly a part of the containment structure of the system. We have simplified the containment modeling technique so that it is limited to relationships that are entirely within the system.

2.5 A Brief Note about Cycles

Upon initial examination, it may seem that simply identifying any container as allowing or disallowing cycles would be sufficient. In some circumstances, however, it is not. For example with ClearCase [7], as in many file systems, directories may contain other directories. This relationship may be cyclic; that is, a directory may, by reference contain another directory. There is nothing to prevent the second directory from also containing the first. On the other hand, some SCM systems express versioning by creating a linear ordering of objects, where each object points to (contains by reference) its successor (or predecessor). This requires the self-referential containment relationship to by acyclic. For SCM systems which version directories this produces a contradiction. The self-containment relationship which indicates that one directory can contain another may be cyclic. The self-containment relationship which represents the linear ordering of versions must be acyclic. Our current cycle property will not accommodate this situation. We leave to future work the development of a richer cycle property (or set of properties) to allow us to correctly model this case.

3 Containment Model Examples

3.1 Hypertext and Hyperbase Systems

Containment modeling allows us to see at a glance the similarities and differences between various systems in a domain. An examination of Figure 4 shows that all hypertext systems center around three main constructs: collections or groups of objects, objects, and connections between objects. We have all used the Web and recognize these general characteristics of hypertext systems. Containment modeling illustrated them clearly.

It also enables us to recognize differences. A key difference between hypertext systems lies in their treatment of links. The Web and HyperCard both embed a set of links in the content of each object as with the "<A href=" construct of HTML. many hypertext systems, however, do not use embedded links: NoteCards, Aquanet, and HyperDisco, for example. Among these systems, though, there is a variation in what the links contain. Links in NoteCards and Aquanet referentially point to a content object, while in HyperDisco, links contain lists of "to" and "from" endpoints. An endpoint is an abstraction that contains an anchor and either a node or a composite. While not a complete survey of data models for linking, it seems reasonable that, having modeled the containment structures of multiple hypertext systems, we could analyze these structures to develop a design space of anchoring and linking.

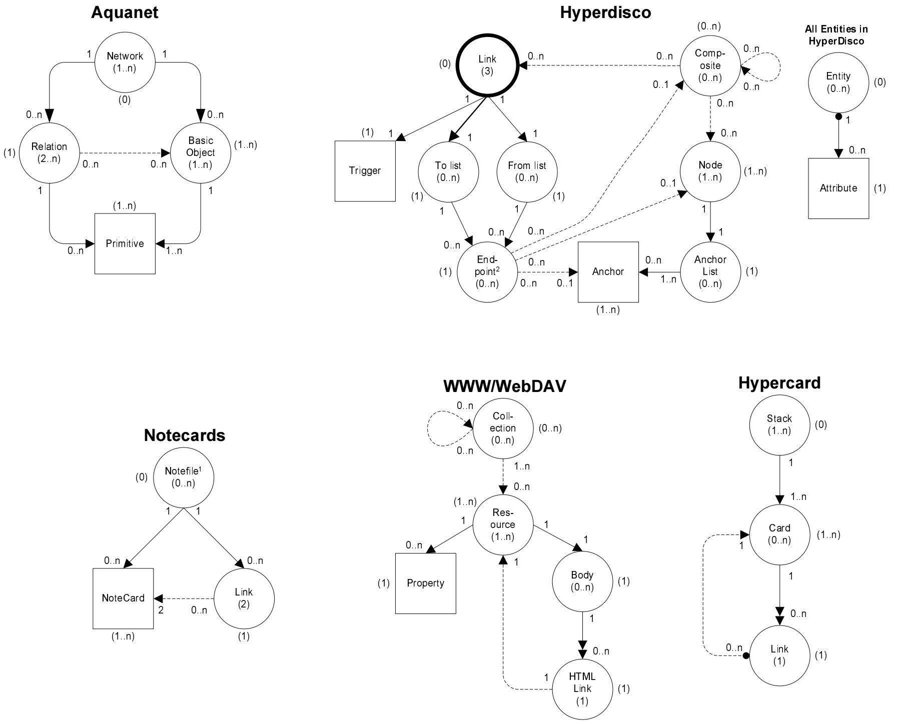


Fig. 4. Containment models of selected hypertext systems: Aquanet [8], HyperCard [1], NoteCards [10], WWW/WebDAV [13], and HyperDisco [14]. Constraints on the diagram are 1 (NoteCards): *contains*(Notefile, Link) → *contains*(Notefile, NoteCard), and 2 (HyperDisco): *contains*(Endpoint, Node) ↔ ¬*contains*(Endpoint, Composite) and *contains*(Endpoint, Node) $\bigvee$ *contains*(Endpoint, Composite) → *contains*(Endpoint, Anchor)

In addition to comparing analogous features of a system, containment modeling can show us overall system characteristics. For example, the containment models in Figure 4 show that both HyperDisco and WWW/WebDAV maintain some metadata. The HyperDisco metadata system is quite extensive. An entity in the system may have attributes associated with it. In addition, links have a particular piece of metadata, a trigger, associated with them. The WWW/WebDAV metadata is less extensive. It is limited to properties on resources. None of the other systems show evidence of metadata collection in their containment models.

As with any model, we must recognize that the information drawn from containment models has its limitations. For example the containment models for Aquanet, HyperCard, and NoteCards show no evidence of metadata. That does not mean that these systems cannot support metadata collection. Any of them could, at a policy level, be used to maintain metadata simply by requiring anyone who adds a new resource to the system to link it to a metadata resource. Similarly, while the containment models for WWW/WebDAV and HyperDisco both illustrate explicit metadata systems, neither system can force users to apply metadata appropriately. For example, keyword searches on the Web often produce totally irrelevant information — because website owners frequently pack their site's keywords with irrelevant words to generate more hits from search engines.

3.2 Configuration Management Systems

As we saw with hypertext systems, containment modeling allows us to look for similarities and differences among systems within a domain.

By examining the models of SCM systems, we can see a fundamental structural difference between PIE and the others. Both ClearCase and Adele depend on collecting all versions of a given item into a single container that represents the abstract notion of each item, independent of a specific revision or variant. In ClearCase this container is the "versioned object," while in Adele it is the "interface," which contains a set of branches representing versions and variants of that interface. The versioned object abstraction is a key enabler for automated assistance in constructing configurations. It allows the system to recognize when two different items in the system represent the same abstraction at different points in time (versions) or space (variants). Versions are recognized because they belong to the same versioned objects. Management of varian and Adele. In ClearCase, variants are explicitly tagged with metadata and are contained within a versioned object. In Adele variants belong to different branches, but to the same interface.

PIE has no analogous structure. This key difference gives PIE users extraordinary flexibility in how they can manipulate and organize their environments, but at the expense of automated support for establishing and maintaining configurations. The more rigorous structure in the other SCM systems provides them with the information necessary to automatically select items to include in a configuration based on higher-level parameters set by the users.

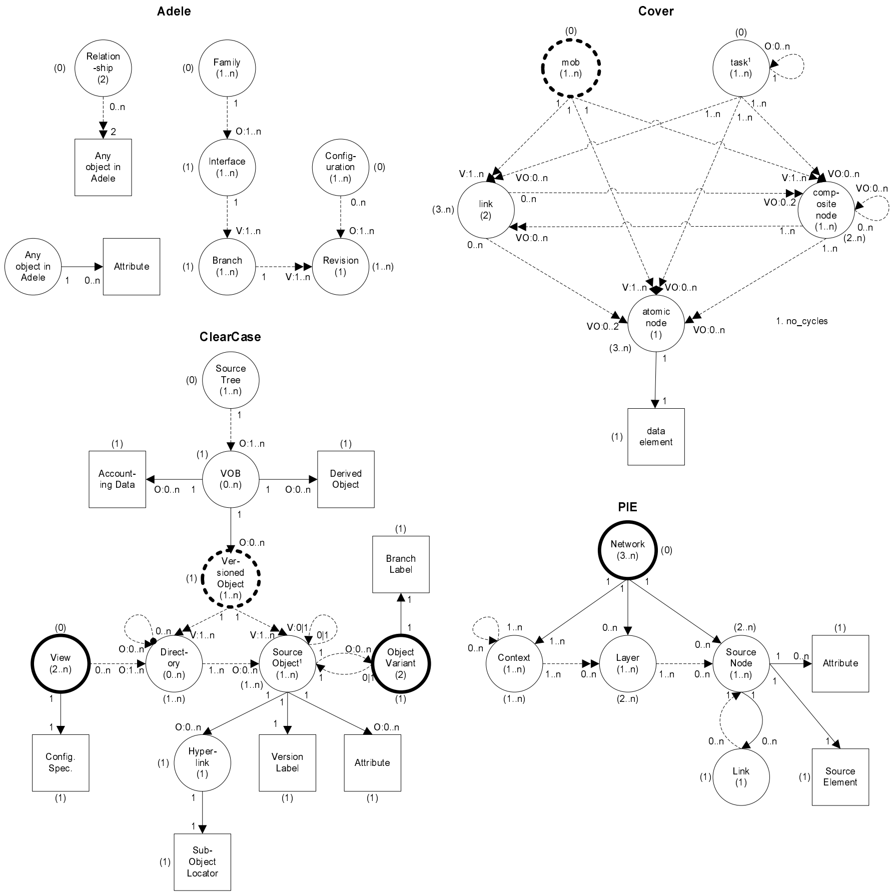


Fig. 5. Containment models of selected configuration management (Adele [2], ClearCase [7], and PIE [3]) and hypertext versioning systems (CoVer [5][4]).

3.3 Cross-Domain Comparisons

In addition to intra-domain analysis, containment modeling can be used to compare systems across domains. We can examine SCM systems and hypertext systems to identify patterns that are common to both. We can also readily identify elements that distinguish the two kinds of systems.

The containment model of CoVer is interesting because it combines aspects of both hypertext systems and SCM systems. Across the center of the model we see a core structure that is common to hypertext systems: a collection of objects and the links that connect them.

At the top and bottom of the model we see structures that are similar to those found in SCM systems. The mob container holds all the versions of a particular object, analogous to the versioned object container in ClearCase. The task container may hold any combination of versions and objects. It fills the workspace niche (called a 'view' in ClearCase).

The SCM containment models show us that designers of SCM systems have also drawn from the hypertext world. All three systems modeled here have link structures. The links even exhibit a similar range of traits to the links we saw in hypertext systems. Some are embedded within specific kinds of entities (as in ClearCase and PIE), and others are independent (as in Adele). In none of the systems are links treated as independently versionable objects.

CoVer is unusual in its treatment of links. We saw examples of independent links in both hypertext and configuration management systems. The independent link entity in CoVer is unusual because it is versioned independently. None of the SCM systems treat links as explicitly versionable. They are versionable in ClearCase and PIE only because they are embedded in other entities that are versioned. Adele does no allow versioning of links at all.

We have seen how containment modeling can highlight similarities between systems in different domains. It can also help determine what makes the domains different. We saw that both hypertext and SCM systems have collections of nodes and the links between them. Hypertext systems stop there. They have three main elements: collections, nodes, and links. While some hypertext systems have more than three entities in their models, the additional entities are either parts of the link implementation, or are entities that represent actual data or metadata. SCM systems, on the other hand, all have at least on more layer. Each SCM system has at least one entity that collects collections. In fact, these meta-collections are roughly parallel to the hypertext collections. The lower level collections (those that collect nodes) are the versioning mechanisms. In ClearCase and Adele the role of these collections is clear: they contain the individual versions of a single, conceptual entity. in PIE and CoVer their role is less explicit since these low-level collections can actually contain any nodes, irrespective of the nodes' relationships to one another. In all four systems, however, these low-level collections are, themselves, collected to construct a complete picture of the system's content.

4 Conclusion

Each system within the SCM and hypertext domains can be viewed as an exploration of a specific point within a multi-faceted space of design choices. Many SCM and hypertext systems have been developed over the past 30 years, providing the raw data that can be analyzed to develop a description of the key decision points in the common design space of content management systems.

The containment modeling technique presented in this paper permits the data models of content management systems to be graphically represented and compared to reveal their commonalities and differences. Containment modeling was applied to a set of nine hypertext, SCM, and hypertext versioning systems, the first time such a broad set of these systems has been compared. Containment modeling can thus be viewed as a meta-modeling technique, in that it provides a mechanism capable of representing the data models of systems in multiple domains.

We intend to use this work as the basis for a thorough domain analysis of content management systems. Containment modeling will allow us to investigate systems in the Hypertext, SCM, Document Management, and VLSI CAD domains, and to identify the key structural components of each. It will help us to distinguish between truly different structures and those that differ only marginally, or only due to terminological differences.

Beyond modeling, we intend to develop a system that can automatically generate content management systems. Given a precise, machine-readable description of a containment model the generator will automatically create a repository consistent with that specification. Automatic generation will provide future researchers and systems builders with a powerful tool for the rapid creation of content management repositories, and the exploration of more complex and powerful data models.

Furthermore, by using the same tool to automatically generate a broad range of content management systems, it will reinforce the commonalities among content management systems.

Acknowledgments

We would like to thank Jim Jones for his generous editing assistance, and Uffe Wiil for clarification of the HyperDisco containment model. Additionally, we thank Jessica Rubart for her detailed thoughts on how to express containment modeling in UML (not all of which were explored in this paper), David Millard for his feedback from using containment modeling to represent the data model of the Open Hypermedia Protocol, and GuoZheng Ge for his detailed review of the PIE containment model. Ongoing work in containment modeling is supported by National Science Foundation grant CAREER CCR-0133991.

References

1. Apple Computer, *HyperCard Script Language Guide*. Reading, MA: Addison-Wesley, 1988.
2. J. Estublier and R. Casallas, "The Adele Configuration Manager," in *Configuration Management*, W.F. Tichy, Ed. Chicester: John Wiley & Sons, 1994, pp. 99-133.
3. I.P. Goldstein and D.P. Bobrow, "A Layered Approach to Software Design," in *Interactive Programming Environments*, D.R. Barstow, H.E. Shrobe, and E. Sandewall, Eds. New York, NY: McGraw-Hill, 1984, pp. 387-413.
4. A. Haake, "Under CoVer: The Implementation of a Contextual Version Server for Hypertext Applications," *Proc. Sixth ACM Conference on Hypertext (ECHT'94)*, Edinburgh, Scotland, Sept. 18-23, 1994, pp. 81-93.
5. A. Haake and J. Haake, "Take CoVer: Exploiting Version Support in Cooperative Systems," *Proc. InterCHI'93 - Human Factors in Computer Systems*, Amsterdam, Netherlands, April, 1993, pp. 406-413.
6. F. Halasz and M. Schwartz, "The Dexter Hypertext Reference Model," *Communications of the ACM*, vol. 37, no. 2 (1994), pp. 30-39.
7. D. Leblang, "The CM Challenge: Configuration Management that Works," in *Configuration Management*, W.F. Tichy, Ed. New York: Wiley, 1994, pp. 1-38.
8. C.C. Marshall, F.G. Halasz, R.A. Rogers, and W.C. Janssen, Jr., "Aquanet: a hypertext tool to hold your knowledge in place," *Proc. Third ACM Conference on Hypertext (Hypertext'91)*, San Antonio, Texas, Dec. 15-18, 1991, pp. 261-275.
9. K. Østerbye and U.K. Wiil, "The Flag Taxonomy of Open Hypermedia Systems," *Proc. Seventh ACM Conference on Hypertext (Hypertext'96)*, Washington, DC, March 16-20, 1996, pp. 129-139.
10. R. Trigg, L. Suchman, and F. Halasz, "Supporting Collaboration in NoteCards," *Proc. Computer Supported Cooperative Work (CSCW'86)*, Austin, Texas, Dec. 3-5, 1986, pp. 147-153.
11. E.J. Whitehead, Jr., "Design Spaces for Link and Structure Versioning," *Proc. Hypertext 2001, The Twelfth ACM Conference on Hypertext and Hypermedia*, Århus, Denmark, August 14-18, 2001, pp. 195-205.
12. E.J. Whitehead, Jr., "Uniform Comparison of Data Models Using Containment Modeling," *Proc. Hypertext 2002, The Thirteenth ACM Conference on Hypertext and Hypermedia*, College Park, MD, June 11-15, 2002, pp. 182-191.
13. E.J. Whitehead, Jr., and Y.Y. Goland, "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," *Proc. Sixth European Conference on Computer Supported Cooperative Work*, Copenhagen, Denmark, Sept. 12-16, 1999, pp. 291-310.
14. U.K. Wiil and J.J. Leggett, "The HyperDisco Approach to Open Hypermedia Systems," *Proc. Seventh ACM Conference on Hypertext (Hypertext'96)*, Washington, DC, March 16-20, 1996, pp. 140-148.