RC23344 (W0409-136) September 21, 2004
Computer Science

IBM Research Report

Concern Modeling in the Concern Manipulation Environment

William Harrison, Harold Ossher, Stanley Sutton Jr., Peri Tarr
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Concern Modeling in the Concern Manipulation Environment

William Harrison, Harold Ossher, Stanley Sutton Jr., Peri Tarr
IBM Thomas J. Watson Research Center, Hawthorne, NY
{harrisn, ossher, suttons, tarr}@us.ibm.com

Abstract

The Concern Manipulation Environment (CME) is an AOSD environment in which software is organized and manipulated in terms of concerns. ConMan supports the identification, definition, encapsulation, extraction and composition of concerns in the CME. ConMan models software in terms of concerns, relationships, constraints, units, artifacts, and associated information. The concern model is multidimensional and concerns can be defined extensionally and/or intensionally. ConMan is neutral with respect to artifact types and formalisms, and it can be used with both aspect-oriented and non-aspect oriented software and methods. ConMan is intended to serve both as a tool for directly modeling concerns and as a platform for developing alternative concern-modeling approaches.

1. Introduction

Concerns are pervasive in software development. They originate in the interests of the stakeholders of a software system, including customers, developers, users. They arise throughout the software life cycle, from ideas for features or functions that inspire new development, through to properties such as maintainability and adaptability that affect the utility and value of a product long after initial development. We recognize many categories of concern, including concerns relating to the physical and logical organization of systems; application-specific concerns such as functions, features; developer-oriented properties such as understandability and maintainability; and user-oriented properties, such as performance and ease of use. Concerns of many different kinds are important across the full range of software engineering activities.

The representation, separation, and integration of concerns are important elements in programming languages, modeling formalisms, and software engineering methods. Nevertheless, many problems with software can still be traced to the limitations of current approaches to managing concerns. These include the *a priori* imposition of particular concerns or types of concern, the failure to treat concerns as first-class entities across the software life cycle, and the imposition of rigid decompositions on software artifacts. Attempts to address these problems have given rise to aspect-oriented software development (AOSD) [13], in which software is fundamentally represented, organized, and manipulated in terms of concerns.

The Concern Manipulation Environment (CME) [12] is an AOSD environment based on the premise that concerns should be treated as first-class entities in all stages of the software life cycle. The CME addresses two main groups of users. For software developers using AOSD, the CME offers tools for creating, manipulating, and evolving aspect-oriented software across the lifecycle. For AOSD tool providers and researchers, the CME offers a flexible, open set of components and frameworks on which to build and integrate aspect-oriented technologies. A key goal of the CME is to promote *incremental adoption* of AOSD. The CME allows developers to identify, model, and analyze concerns in existing software, regardless of whether it was developed using AOSD.

Any software environment that organizes and operates on software according to concerns requires support for concern modeling. This paper describes ConMan, the Concern Manager component of the CME. Section 2 introduces the topic of concern modeling and gives some scenarios of concern modeling in software development. Section 3 states our requirements for a concern-modeling facility. Section 4 then presents the ConMan schema and addresses issues of implementation. Section 5 discusses ConMan in the context of the CME and Eclipse [12], and Section 6 describes our experience with it. The final sections address related work and summarize our results.

2. Concern Modeling

Concern modeling is simply the representation of concerns themselves as first-class entities. The following scenarios suggest how concern modeling might be used in a variety of development scenarios:

In a **COTS-based process**[1], a concern model of the system under development can be constructed to provide a framework for evaluating the compatibility and contribution of COTS products that are candidates for adoption. The suitability of candidate products can be evaluated based on the range and compatibility of concerns addressed, the need for tailoring, and the number of concerns not addressed.

In **new development**, a concern model can be elaborated to represent the initial concerns that motivate or constrain the project. As development progresses the concern model can be elaborated. Concerns can be related to the work products in which they are addressed, and relationships between concerns can be drawn to capture semantic and other dependencies. The concern model can be analyzed for consistency and coherence. As the life cycle is iterated, changes at various stages can be validated against the concern model, and the model can help to propagate updates through both the concern space and the artifact space.

In the **retroactive development of a product line** based on an existing product, concern modeling can be used to characterize the potential feature space and to position different product variants in that space. The existing product can be decomposed into units corresponding to specific concerns. Concerns representing additional features can be identified and related to the existing model, and additional work products can be developed to support implementation of the additional concerns. Variants within the family can be specified by selecting concerns of interest from within the family concern space. Compositional technologies can then be used to compose variants using implementations associated to the selected concerns.

We could elaborate many additional scenarios, for example, relating to software understanding**, extension, deployment, and multi-team development**. Concern modeling thus has a role in many kinds of development tasks and processes.

3. Concern-Modeling Requirements

While many of the requirements for concern modeling are to be expected of any meta-modeling facility, a good concern modeling tool must address several important modeling requirements that are less common: specific modeling concepts for *concerns* and their organization, neutrality and open-endedness with respect to artifacts, and specification that captures the intended structure of material rather than simply reflecting existing structure.

As part of the CME, the tool itself faces several noteworthy requirements: it should be able to deal with artifacts across the whole lifecycle, with software creation as well as with result artifacts, and with material that is often incomplete and incorrect as it is being developed. It also requires a modern interactive interface to developers that have to work with the model

3.1. Schema Requirements

The concern-modeling schema must be "concern neutral", that is, able to capture **arbitrary concerns**. This is necessary to represent concerns for all kinds of stakeholders and all sorts of development tasks. To represent arbitrary concerns it is necessary to represent both **abstract and concrete concerns**. Abstract concerns include conceptual entities, such as features in the abstract, properties, topics of interest, and so on. Concrete concerns represent physical entities, notably the work products of software development.

The concern modeling schema must support multiple, concurrent, overlapping *classifications* or *dimensions o*f concerns. Concern spaces are certainly **multidimensional** [30], that is, concerns are typically organized according to multiple classifications. For example, a particular class in an object-oriented application might be classified according to stakeholder relevance, development stage, features, complexity, size, and more, each of which may be taken as a dimension of concern. We must be able to support **multiple levels** of concern, representing arbitrary levels of abstraction or detail. These may include multiple levels of classification, structure, aggregation, and so on. For example, functionality may be organized by feature by specific functions, and by functional variants (within functions). It should be possible to form arbitrary groups of concerns or other model elements, on demand. This supports the flexible formulation of concerns and also allows for "working sets" of model elements that may needed during concern-modeling activities

It is necessary to represent various types of artifacts, but our schema must be **neutral with respect to the kinds of artifacts** that can be modeled. This is to be able to capture artifacts of interest to the whole range of stakeholders, from across the life cycle. Similarly, our schema should be **neutral with respect to the formalisms** used to represent particular types of artifacts. This is important for capturing artifacts from across the life cycle and across different development methods. Our schema should also be **neutral with respect to the development approach**. In particular,

[1] That is, a process using commercial, "off-the-shelf" software

it should be useful not only within AOSD methods but also in more traditional development methods. As a particular case of method neutrality, the schema should be **symmetry-neutral**: useful in a variety of aspect-oriented development methods and tasks, including both those with distinguished aspects and bases (asymmetric models) and those without distinguished aspects and bases (symmetric models).

It is not sufficient to represent concerns in isolation; we must also **represent relationships** among concerns. These arise from many sources, including the semantic properties of artifact types and instances, dependencies in the development process, characteristics of the application domain, and stakeholder world-views. Relationships among concerns are important because they contribute directly to an understanding of software, help to organize development tasks, and constitute concerns themselves. For broad applicability, we must represent relationships of arbitrary kinds, among arbitrary types of elements. Moreover, relationships should be first-class entities in the concern model. To represent arbitrary levels of abstraction or detail, we want to be able to aggregate detailed relationships over lower-level elements and represent them as more general relationships between higher-level elements. **Constraints** on concerns arise from many sources, including domain models, customer requirements, methodology, implementation contingencies, and more. Constraints are important because they define consistent states of models and assure that concerns are used consistently in development. Constraints, like relationships, should be first-class entities.

It should be possible to define a wide range of concern model elements, including concerns, relationships, and constraints, by either *extensional or intensional specification* (or both). This affords flexibility and simplicity in building and maintaining concern models. To support intensional specifications, a notation for writing the specifications is also required. Much concern modeling will directly involve software artifacts and information or elements they contain. Although we want a concern-modeling schema that is artifact-neutral, it will often be helpful to be able to capture **artifact-specific information**, although in ways that are independent of the artifact formalism. It will also be useful to attach **arbitrary information** to model elements. This may be information that is not essential to the concern model itself but is useful in some particular context, such as determining how a model should be displayed or processed. In order to support not just concern modeling but also the development of additional concern-based or aspect-oriented tools and methods, our schema should be **open and extensible**. Some points of openness have already been indicated, for example, openness to types and instances of concerns, and the need to accommodate artifact-related and general kinds of information. The schema should also provide a framework for the definition of additional kinds of modeling entities.

We intend our concern modeling schema to be used directly to model concerns. However, we recognize that there are other approaches to concern modeling (for instance, [28], [22], [34]), and we hope to promote development and experimentation with varied modeling approaches, tools and methods. Thus, we want a concern modeling schema that can serve as a unifying basis or **metamodel** for defining alternative and higher-level concern modeling schemas.

3.2. Component Requirements

The basic requirement for a concern-modeling component for CME is to support concern modeling according to a schema that addresses the requirements outlined above. Additionally, there are important requirements beyond support for the schema..

Because we expect concern modeling to occur in many different phases and in many different contexts, a concern-modeling component must be able to accommodate incremental and incomplete modeling. Although the concern-modeling schema must allow concerns to be modeled on many different levels, a concern-modeling component must allow users (human or automated) to collapse or expand the view and to access the model on any of the different levels and to move between levels. And because we plan to apply concern modeling to large-scale, real-world systems, a concern-modeling component must perform well and scale to large bodies of software.

We view concerns as constituting an extensive and highly structured space, and a concern-modeling component should support users in locating themselves in and navigating through that space. As a means of navigating through a concern space, focusing user attention, obtaining information about a concern or space, and also for defining concerns, a concern-modeling component should support queries over the space.

Supporting the incremental adoption of AOSD is one of the principal goals of the CME. Toward that end, it should be possible to use a concern-modeling component without requiring the use of other aspect-oriented technologies. There are additional properties that are of interest to us for a concern-modeling component that we expect to address mainly in subsequent work. These include incrementality, concurrency, inconsistency management, and others.

4. ConMan Schema and Component

Here we describe the main elements of the ConMan schema and discuss implementation issues.

4.1. Schema Design

At the topmost level a ConMan model consists of a *concern space*, which contains elements representing concerns, relationships, constraints, and units.[2] Examples are shown in Figure 1, which represents the Concern Explorer view from the CME (an Eclipse plugin).

ConMan *concern*s represent the conventional notion of a concern and can contain other *concern model elements* (such as nested *concerns* or *units*). *Concern contexts* (which specialize *concern*) can also have associated *relationships* and *constraints* (which are maintained separately from any *relationships* or *constraints* that the concern may include as elements). Thus the structure of a *concern space* can be built up both by the grouping of model elements and by explicit *relationships* and *constraints* among them. In Figure 1, "Workspace", "example.cme", "Features", "Naming", "Error types" are all examples of *concerns*. "Workspace" is a *concern context*, and its associated *relationships* are found in the folder "ContainedRelationships."

Concerns can be assigned elements extensionally, intensionally, or by a combination of these approaches. Extensionally defined *concerns* (such as "Printing") are assigned their elements directly. Intensionally defined *concerns* (such as "Naming") obtain their elements through evaluation of an associated query (and users can specify policies as to when these queires are evaluated).. A particular subtype of *concern context* is *composition*, which contains a set of elements to be composed and a set of composition relationships that describe how to compose them.

Relationships in ConMan represent relationships among *concern model elements* of various types. *Relationships* are typed in the ConMan schema by their structure and access methods. ConMan *relationships* can be binary or n-ary, and the elements in them can be ordered or not. *Relationships* with positioned elements can be considered directed, *relationships* with named elements can be considered directionless. In implementation, ConMan *relationships* are not typed by relationship semantics; so, for example, directed binary relationships are used to represent many kinds of semantics, such as *implements*, *extends*, *refersTo*, and others among Java artifacts. To allow for the representation of such semantics, ConMan also supports specialized *attributes* on *relationships*. In Figure 1 relationships are shown for the Java interface org.eclipse.cme.Entity.

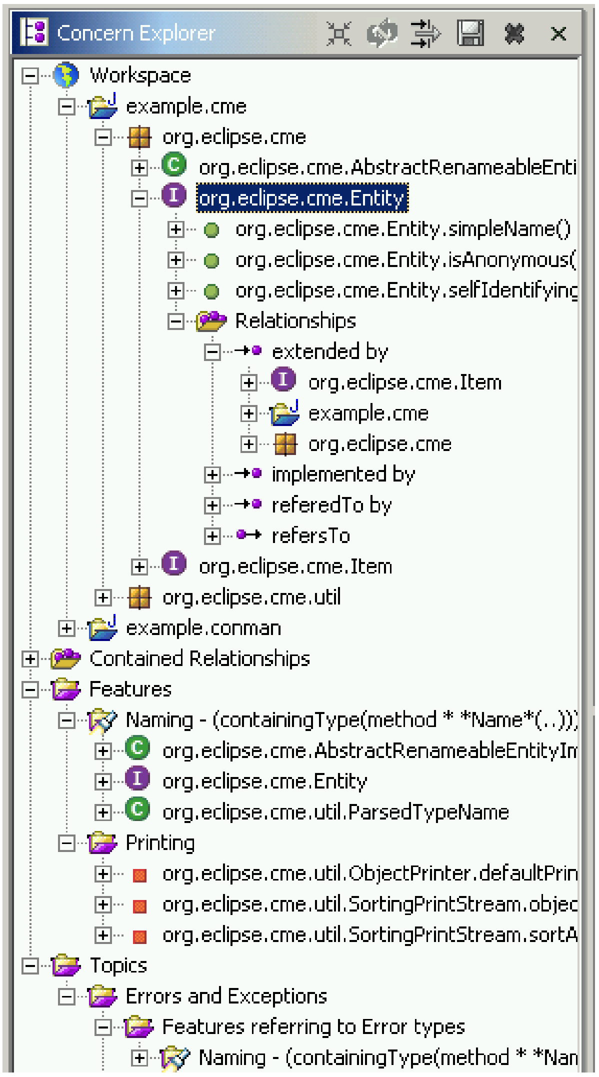
Concern Explorer
Workspace
example.cme
org.eclipse.cme
org.eclipse.cme.AbstractRenameableEnti
org.eclipse.cme.Entity
org.eclipse.cme.Entity.simpleName()
org.eclipse.cme.Entity.isAnonymous(
org.eclipse.cme.Entity.selfIdentifying
Relationships
extended by
org.eclipse.cme.Item
example.cme
org.eclipse.cme
implemented by
referedTo by
refersTo
org.eclipse.cme.Item
org.eclipse.cme.util
example.conman
Contained Relationships
Features
Naming - (containingType(method * *Name*(..)))
org.eclipse.cme.AbstractRenameableEntityIn
org.eclipse.cme.Entity
org.eclipse.cme.util.ParsedTypeName
Printing
org.eclipse.cme.util.ObjectPrinter.defaultPrin
org.eclipse.cme.util.SortingPrintStream.objec
org.eclipse.cme.util.SortingPrintStream.sortA
Topics
Errors and Exceptions
Features referring to Error types
Naming - (containingType(method * *Nan

Figure 1. The CME Concern Explorer View

Whereas *relationships* group related elements, *constraints* group constrained elements. *Constraints* can also be added to *concern contexts* in a distinguished role (that is, as constraints associated to the context, not as elements of the context). ConMan does not yet have a constraint definition language, but Java can be used to provide an implementation of constraint semantics, as can patterns in the CME query language.

Concerns, *relationships*, and *constraints* are all specializations of *concern model element*, the basic type of ConMan models. As such, all have first-class

[2] In this section we use italic font to distinguish references to modeling concepts in the ConMan schema.

status, and all can be applied to one another. Consequently, any *concern model element* can be loaded into a c*oncern space*, grouped into a *concern*, related by a *relationship*, or constrained by a *constraint*. These elements also support multidimensional separation of concerns MDSOC and crosscutting associations. For example, in Figure 1 the feature "Naming" occurs under both the "Features" *concern* and also under a "Topic" concern o which it is related. *Units* (discussed below) can also crosscut multiple *concerns*, *relationships*, and *constraints* in a *concern space*, such as the interface org.eclipse.cme.Entity in Figure 1.

The elements described so far support concern modeling in the abstract, but concerns will often be related to artifacts that represent the work products of development. To represent these, ConMan uses *units* and *artifacts. Units* are *concern model elements* whose purpose is to represent, in the concern model, the work products of software development (or other sorts of documents, models, and so on). *Units* may be simple or compound. In Figure 1, compound *units* are used to represent classes and interfaces, simple *units* to represent fields and methods. A *unit* has a *definition*, which is a reference to an *artifact*. *Artifact*s are not *concern model elements* as such but represent (either directly or by proxy) the entities that are modeled as *units*. An *artifact* has a *location* that indicates where the actual artifact represented by the *unit* can be found. *Artifacts* may represent many sorts of things, including files in a file system, distinguished elements of files, elements of other models (including in-memory models), elements stored in databases, and so on.

Although ConMan is intended to be artifact neutral, many of the artifacts that will be modeled during software development will come from formalisms that have rich semantics. To enable some of that information to be captured in a ConMan model, elements can have associated *modifiers classifiers*, and *attributes.*

Modifiers represent terms in an artifact formalism that play a qualifying role, such as visibility-controlling keywords. *Classifiers* are terms that represent kinds artifact in the artifact formalism (e. g, classes in Java). For example, the Concern Viewer (Figure 1) uses *classifiers* to discriminate Java classes and interfaces, and it uses *modifiers* to distinguish public from private members. Thus, specific sets of *modifiers* and *classifiers* are formalism-specific (although some may be shared by multiple formalisms), but the notions of *unit* and *artifact* and the modifier and classifier mechanisms are generic. ConMan also has a general-purpose *attribute* mechanism that allows arbitrary information to be associated with any element.

A concern space also has associated *loaders* and *builders*. Loaders are responsible for loading elements into a concern model from some source domain (typically software artifacts or other models). Loaded elements may include concerns, relationships, constraints, units, and so on. Builders extract information from a concern model for use by tools (possibly outside of ConMan) in building new concern model elements (e.g., in composing new artifacts). Detailed discussion of loaders and builders will appear in later work.

The ConMan schema includes a number of open points that support our goal that the CME should be a platform for development of new aspect-oriented tools and approaches. The attribute, modifier, and classifier mechanisms allows users to associate arbitrary information with *concern model elements*. The ConMan schema can also be used as a framework for the definition of new types of *concern model element*. *Loaders* and *builders* also constitute extensible frameworks. Examples of specific extensions are discussed in Section 6. Of course, the specific types and instances of concerns, relationships, and constraints in a concern space, and the structure of the space, are also open.

Additionally, ConMan works with other elements in the CME. One of these is Panther, a query manager that supports the definition of intensional concerns in ConMan. Another is CIT, the Concern Informant Toolkit, which provides a standardized, abstract interface for artifacts and their elements. CIT enables ConMan (and its loaders and builders) to access a variety of types of artifacts in a uniform way. It provides a generic object-oriented interface that can be implemented by specific object-oriented types and also other types that are based on a container-element model.

We believe that the ConMan schema, as designed and implemented, generally addresses the requirements set out in Section 3 and affords great flexibility in the modeling of concern spaces. Some experience that helps to bear this out is described in Section 5.

4.2. Operational Issues

One of our operational requirements was to support incremental and incomplete concern modeling. This is readily accommodated by the ConMan schema, which has no *a priori* completeness requirements. It is also accommodated by the loader model and implementation, which support the loading of individual elements and small sets of elements. To further address this objective, we have introduced into *compound_unit* implementations the ability to lazily load the details of a unit as needed. We also support the loading of different categories of relationship at different times.

Performance and scalability are critical to the adoption of concern modeling for large, real world systems. We have addressed these issues in several ways. One is through incremental loading of concern models as-needed. Another is by the use of different means of accessing the artifacts from which a model is built. For instance, for accessing Java artifacts, we have made use of both JikesBT [18] and Shrike [26], which offer different performance and space characteristics. The implementations of grouping data structures can also be optimized for space or access. Additionally, we are pursuing space optimizations in the ConMan implementation and performance optimizations in the loader architecture and algorithms. As most of our implementations represent evolving prototypes, it is not possible at this time to give representative size or performance figures; however, an early ConMan implementation was able to provide practical support for the simple concern modeling of a system of in excess of 15,000 Java classes (Section 6.1 and [8]).

A typical software system contains many kinds of element with many kinds of relationship. For Java systems, we compute over a dozen relationships among code units alone. In order to simplify the view of relationships, and to represent them at higher levels of granularity, our relationship loaders for ConMan support the propagation of relationships from lower-level (contained) elements up to higher-level (containing) elements, summarizing at higher-levels of granularity the relationships at lower-levels of granularity. For example, if two classes are related by some relationship (e.g., "extends"), that relationship is propagated to containing concerns that represent, for example, packages, projects, components and features.

ConMan has both a GUI and an API. The API represents the kinds of schema elements described in Section 4.1 and is used by the loaders and other tools. Elements of the GUI are described next.

5. ConMan in the CME and Eclipse

ConMan is represented in the Eclipse plug-in for CME through perspectives that combine a number of views related to concern modeling and use.

The **Concern Explorer view** (Figure 1) displays a ConMan concern model in a tree-structured form (although the models can have more general structures). The default loader for ConMan models of Eclipse workspaces creates three top-level elements. One of these represents the workspace itself, with concerns and units under it representing the folders, packages, and files of the workspace. These are, after all, key concerns in the software. The model has an organization parallel to that of the workspace and a representation in the GUI like that used in the Eclipse Package Explorer view. An addition that we make to the representation of the workspace is to add the "Relationships" folder for each unit represented.

Another top-level element in the default model is a folder for "Contained Relationships" in the concern space; by default these are initialized to hold all of the relationships among the Concerns and Units in the workspace. These include summary relationships, that is, relationships aggregated from lower-level elements to higher-level elements (as shown by the "extended by" relationships for org.eclipse.cme.Entity).

The third top-level element, the "Features" concern, is initially empty and represents a default location for users to add feature-related concerns through the UI (as we have done in Figure 1). Other top-level concerns can also be added (such as "Topics" in Figure 1).

The **CME Search and Search Results views** support a concern-oriented search capability based on a pattern-matching query language [29]. Searches can be executed over the entire concern model, the workspace, or specific concerns, and they can be focused on specific kinds of concern-model elements. The results of query evaluation are displayed in a Search Results view. Query results can also be sent to a concern and used to define a concern extensionally. Alternatively, the query itself can be sent to a concern and used to define the concern intensionally.

Finally, the CME has a **Visualizer view** that shows overviews of workspace elements, highlighting parts identified by query. This is useful for showing the distribution of concerns across units.

6. Experience

In this section we discuss three sorts of experience with ConMan: The modeling of a large software base for purposes of concern extraction; the implementation of another concern-modeling schema; and extensions to the ConMan schema and loader framework to accommodate new types of artifacts in concern models.

6.1. Extracting a Concern

As reported in [9], our colleagues at IBM Hursley Park (supported by the authors) used CME capabilities to conduct an experiment in concern extraction. The goal was to take a large application server (over 15,000 classes) and extract the components relating to the use of Enterprise Java Beans (EJB) [27], thus forming a small product line in which the application server could be deployed with or without EJB support.

ConMan was used to defined a concern model including the EJB container concern and concerns for each of about 80 components. The EJB container concern was initially populated with components that implemented the EJB support. An iterative process guided by queries over the ConMan model was used to identify and eliminate dependencies between other components and components in the EJB concern. This often involved refactoring, sometimes creating AspectJ [20] aspects, and sometimes putting additional code into the EJB concern. The separated components were subsequently recombined using AspectJ through the Eclipse AspectJ Development Toolkit [2].

The experiment was considered a success overall. Although the tools used needed refinement, the authors in [9] were happy to recommend the cautious adoption of AOSD in commercial projects.

6.2. Modeling an Alternative Schema

Like ConMan, Cosmos [28] is a "next generation" concern modeling schema. Like ConMan, Cosmos was designed to overcome limitations in the Hyperspaces model [17], supports symmetrical modeling, supports multidimensional separation of concerns (MDSOC) [30], and uses notions of concern, relationship, and predicate (or constraint). Nevertheless, there are some significant differences between them.

One important difference is that Cosmos has mainly "flat" elements (lacking nesting) but uses relationships for structuring, whereas ConMan has mainly structured elements (with nesting) although it also including relationships. Another is that Cosmos has more elements with more specific semantics whereas ConMan has more elements with more generic semantics. For example, Cosmos includes metamodeling concepts (e.g., concern types for classifications, classes, and instances) whereas ConMan has the more generic and less strongly typed *classifier* mechanism. Cosmos also has a richer model of physical entities.

On the other hand, Cosmos omits intensional definitions for concerns whereas ConMan includes them, Cosmos has fixed sets of attributes whereas ConMan has several flexible attribute mechanisms, and ConMan has a greater variety of grouping mechanisms.

These differences are attributable mainly to the fact that Cosmos commits to specific concern modeling semantics whereas ConMan is intended to be a more general platform, able to support concern modeling directly and alternative modeling approaches.

To test the latter goal, we have experimented with modeling the Cosmos schema and models in ConMan. One approach is to make use of explicit containment (i.e., grouping) in ConMan to model certain sorts of relationships among elements in Cosmos. This works well for modeling the Cosmos schema, which comprises a near-hierarchical system of categories: ConMan concerns representing higher-level Cosmos categories (such as "logical concerns") can contain ConMan concerns representing Cosmos subcategories (such as "logical topics"). It was also possible to build specific Cosmos models using this technique. In this case, containment serves to represent the typing of specific model elements. However, if containment of elements is used to represent subtyping, then (to avoid ambiguity) it should not be used to represent other sorts of relationships among schema elements (for example, the association of a Cosmos concern to a "logical topic"). For these other sorts of Cosmos relationships we were able use ConMan relationships.

A limitation of this approach is that the typing of Cosmos Elements is not replicated by containment (grouping) relationships among corresponding elements in the ConMan realization. For instance, certain types of Cosmos elements can only participate in certain types of Cosmos relationships (such as Cosmos "logical classes in Cosmos "generalization relationships"). Such restrictions cannot be achieved by containment. However, the desired semantics can be attained through the use of ConMan constraints.

An alternative approach to this problem is to extend the ConMan schema with new types to represent Cosmos elements (e.g., one subtype of Conman concern to represent Cosmos "logical concerns" and another to represent "physical concerns"). This creates a closer semantic match between the Cosmos model and the ConMan realization. However, a complete match is not possible by specialization alone, as Cosmos semantics require covariant specialization, which is not supported in Java. Thus, constraints still must be used.

6.3. Extending the Schema

The initial CME and ConMan implementations focused specifically on Java-related concerns, artifacts, and relationships. However, the architecture of the CME is intended to be open and extensible in multiple dimensions. With respect to ConMan and related elements, we have taken advantage of this in several ways.

The original artifact types were supplemented by support for Ant [1]. Ant is a language for describing software builds in XML [33]. Because Ant artifacts are an integral part of a body of software, it is desirable to include them in the concern model. This was simply accomplished by building a small Ant loader, extended from ConMan's original Java loader, in about two person-weeks. This loader added the artifacts, entities, and relationships needed to the concern model. Types of relationship and artifact appropriate for Ant were defined by extension from ConMan's core types.

Support for AspectJ [10] was also added to the CME. The AspectJ loader was implemented by providing a "concern informant component" [15] for AspectJ that enabled AspectJ artifacts (aspects, pointcuts, etc.) to be loaded with existing "Java" loaders that also use the generic concern-informant interface. Types for AspectJ elements were defined by extension from existing ConMan units and artifacts.

The original ConMan loader implementations were relatively *ad hoc*. Subsequently, a more general, extensible loader framework was designed that has been instantiated for several artifact models, including the Eclipse core resource model, the Eclipse JDT resource model, the CME Concern Informant Toolkit (CIT) [16] type spaces, and files and directories based on java.io.File. The time required to instantiate each of these of loaders varied from two to five person-days. Loaders representing combinations of these models have also been instantiated. The introduction of these models into ConMan also entailed just limited extension to ConMan Artifacts and Relationships.

Additionally, as part of the implementation of a Conman schema and loaders for Cosmos, some extensions to ConMan concerns and units were made comparable to those made for AspectJ support.

7. Related Work

Much work in non-aspect oriented modeling addresses concerns in particular life cycle stages (although some spans multiple stages). Examples include (from requirements) i* and Tropos [35], [7] and KAOS [5], and (from architecture) Attribute-Based Architectural Styles [21] and Domain-Specific Software Architectures [32]. In ConMan these play the role of "artifact formalisms"; they address specific types of concerns but do not treat concerns in general as first-class.

A more general category of non-aspect oriented modeling approaches is represented by UML [24] and comparable "generic" modeling formalisms. These don't overtly restrict the kind of concern that can be addressed, or even necessarily the "meta level" of model, but they provide a specific kinds of constructs for performing modeling (such as packages, classes, associations, attributes, and so on). When such formalisms are applied to software development, the kinds of things modeled can readily be seen as addressing specific kinds of concern (indeed, the various kinds of diagrams in a formalism like UML can be seen as defining various dimensions of concern). However, the constructs in these models more general-purpose than those in ConMan, which embodies notions particular to the domain of concern modeling (such as concern space, concern, and composition relationships) that are absent from more general modeling formalisms.

Another perspective on generic software modeling is represented by "feature modeling." Features are often considered to represent some distinguished, often user visible function or property of a system [19][23], and we often model "features" as one dimension of concern in a space. However, the concept can be expanded to encompass anything of interest in a system [11] (although this extreme view is not typical in practice). Features are commonly modeled using concepts similar to those we propose for concern modeling: aggregation, decomposition, generalization , specialization, parametrization, constraints, relationships. and so on. Thus we believe that concern modeling as we propose it can support feature modeling. (Although ConMan itself does not address parameterization, concerns can be defined based on parameterized queries.) Additionally, concern modeling can complement feature modeling by relating feature modeling and models to other development activities and artifacts.

With the rising popularity of aspect-oriented software development there has been a flurry of work in aspect-oriented modeling. Some of this addresses specific phases of the life cycle, including aspect-oriented requirements engineering [1], [6] architectural analysis [4], [31], and design [8], as well as in the area of more general concern modeling. In contrast to the modeling approaches reviewed above, a major goal of these recent approaches is to identify concerns (or aspects) as such, bringing aspect-orientation into the formalisms used for particular development stages or artifacts.

Concern modeling in a still more general sense is now addressed by several approaches. ConMan is a generalization of the Hyperspaces approach first proposed in [31] and implemented in Hyper/J [17]. Hyper/J used a multidimensional concern model but one that was flat (in that concerns could not be nested) and lacked support for relationships and constraints other than related to artifact composition. ConMan accommodates hierarchical concern models and incorporates relationships and constraints as first-class elements. Wagelaar [34] proposed a concept-based approach called CoCompose for the modeling of early aspects. In CoCompose the concepts involved in a software system are first modeled independently of any implementation; the conceptual models can then be processed to automatically generate an implementation. Lohman and Ebert [22] also proposed a generalization of Hyperspaces [17] that replaces orthogonal dimensions of concerns with non-orthogonal clusters of concerns and allows a unit to be assigned to more than one concern in a dimension. Lohman and Ebert also distinguish primary and secondary dimensions in which the primary dimensions are based on artifacts and the secondary dimensions represent user interests that are not derived from corresponding artifacts. ConMan, in contrast to Hyper/J, does not require concerns to be orthogonal and does not restrict a unit to being in only one concern in a dimension. ConMan also supports the uniform modeling of both artifact-based and artifact-independent concerns from throughout the life cycle, although it does not require it. Thus, we believe that ConMan can support the specific approaches proposed in both [34] and [22].

FEAT [25] [14] is a tool for locating, describing, and analyzing concerns in source code. It contains three components, one of which is a model component, more or less analogous to ConMan. Like the CME, FEAT can operate as an Eclipse plug-in and also interoperates with other Eclipse views. A concern in FEAT is a container for an element of source code that represents a class, member, or relationship of interest. These can be identified by iteratively querying the source code. Queries comprise a set of analyses over the structure of a program. FEAT can also analyze dependencies among elements in concerns, although it lacks constraints. Thus, FEAT contains a set of capabilities that are similar in spirit and partly overlapping with those in ConMan and the CME. However, FEAT is more strictly code-focused. For example, FEAT concerns cannot nest, and only relationships in code are addressed. Also, FEAT lacks many of the open points of CME and ConMan. FEAT does support a richer set of program analyses than CME does. Thus the strengths of FEAT and ConMan/CME are somewhat complementary, and it would be exciting to have FEAT capabilities integrated with ConMan and CME.

Cosmos [28] is another extension of the Hyperspaces model. It overcomes some of the limitations of that model in ways that are similar to ConMan, but there are also significant differences between the approaches. Cosmos is discussed in Section 6.2.

8. Summary

In AOSD software is represented, organized, and manipulated in terms of concerns. The Concern Manipulation Environment is an AOSD environment that treats concerns as first-class entities across the life cycle. ConMan is the Concern Manager of the CME.

ConMan addresses an extensive set of requirements, including support for arbitrary concerns, support for abstract and concrete concerns, modeling of multidimensional and multilevel concern spaces, support for software units and artifacts, neutrality with respect to artifact types, artifact formalisms, and development approach, the ability to support symmetric and asymmetric modeling, and the ability to represent relationships and constraints. ConMan also allows concerns to be defined extensionally or intensionally and to have associated information representing generic attributes or formalism-specific modifiers and classifiers. ConMan is also intended to serve both as a tool for directly modeling concerns and as a platform for developing alternative concern-modeling approaches.

The modeling concepts in ConMan speak directly to these requirements and include concern spaces, concerns, relationships, constraints, units, artifacts, groups, and others. These are generally first-class notions, enabling the construction of highly structured and semantically rich concern models. Additionally, the implementation architecture of ConMan is open and extensible in several dimensions. In the CME, ConMan works with a number of other components and capabilities, including the query engine, visualization capabilities, and composition and extraction mechanisms.

We have used ConMan in a number of validation exercises. These include the extraction of a complex concern from a large body of software, the representation of an alternative concern-modeling schema, and extension to accommodate new types of artifacts, all of which were accomplished with substantial success.

AOSD holds unique promise for addressing persistent problems in software engineering. We believe that concern modeling is at the heart of AOSD and that it can also help significantly with "traditional" software development. With ConMan we have identified and addressed key requirements for concern modeling capabilities. Thus ConMan can support concern modeling in a wide variety of development scenarios and can serve as a platform for further research in this area.

Acknowledgments

We thank Matt Chapman, Andy Clement, Helen Hawkins, and Sian January for their work in implementing CME components and Matt and Andy for extending the components to AspectJ and Ant. We thank Andy and Adrian Colyer for their recommendations regarding concern modeling and their early use of Conman. We are also grateful to Juri Memmert for his comments on concern modeling, as well as to Isabelle Rouvellou for her help in formulating Cosmos.

References

[1] Apache Ant, http://ant.apache.org/

[2] AJDT: AspectJ Development Tools Eclipse Technology Project. http://www.eclipse.org/ajdt/.

[3] E. Baniassad and S. Clarke, "Finding aspects in requirements with Theme/Doc", In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*; at 3rd Int'l Conf. on Aspect-Oriented Soft. Dev., 2004. http://trese.cs.utwente.nl/ workshops/early-aspects-2004/Papers/Baniassad-Clarke.pdf

[4] L. Bass, M. Klein, and L. Northrop, "Identifying aspects using architectural reasoning". In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*; at 3rd Int'l Conf. on Aspect-Oriented Soft. Dev., 2004. .http://trese.cs.utwente.nl/ workshops /early-aspects-2004/Papers/BassEtAl.pdf

[5] P. Bertrand, R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, "GRAIL/KAOS: An environment for goal driven requirements engineering", *Research Handout, 20th Int'l Conf. Soft. Eng*., 1998.

[6] I. Brito, and A. Moreira, "Integrating the NFR framework in a RE model", In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*; at 3rd Int'l Conf. on Aspect-Oriented Soft. Dev., 2004. .http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/BritoMoreira.pdf

[7] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: The Tropos project", *Inf. Sys. 27*, 6, 365–389, 2002.

[8] S. Clarke, W. Harrison, H. Ossher, and P. Tarr, "Subject-oriented design: Towards improved alignment of requirements, design and code", In *14th Conf. Object-oriented Programming, Systems, Languages, and Applications*. ACM, 325–339, 1999.

[9] A. Colyer and A. Clement, "Large-scale AOSD for Middleware." In *Proceedings of the 3rd Int'l Conf. on Aspect-Oriented Soft. Dev.,* pp. 56—65, 2004.

[10] A. Colyer, A. Clement, G. Harley and M. Webster. Eclipse AspectJ, Addison-Wesley, 2004.

[11] K. Czarnecki and U. Eisenecker, "Generative Programming—Methods, Tools, and Applications", Addison-Wesley Professional, 864 p., 2000.

[12] Eclipse.org. "Concern Manipulation Environment Project", http://www.eclipse.org/cme

[13] T. Elrad, R. Filman, and A. Bader, "Aspect-oriented programming". *Comm. ACM 44*, 10, 29–32, 2001.

[14] FEAT, http://www.cs.ubc.ca/labs/spl/projects/feat/

[15] W. Harrison, H. Ossher and P. Tarr, "Concepts for Describing Composition of Software Artifacts." Submitted.

[16] W. Harrison, H. Ossher, P. Tarr, V. Kruskal, and F. Tip, "CAT: A Toolkit for Assembling Concerns" IBM Research Report RC22686, December, 2002

[17] IBM Corp. http://www.research.ibm.com/hyperspace/.

[18] JikesBT, dev.eclipse.org/viewcvs/indextech.cgi/ org.eclipse.cme/contributions/jikesbt/

[19] K. Kang, S. Cohen, J. Hess, W., Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study",. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M Loingtier, and J. Irwin, "Aspect-oriented programming", In *ECOOP'97 Object-Oriented Programming, 11th European Conf.,* M. Akşit and S. Matsuoka, Eds. LNCS, vol. 1241. Springer-Verlag, 220–242, 1997.

[21] M. Klein and R. Kazman, "Attribute-based architectural styles", Tech. Rep. CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon Univ.. Oct., 1999.

[22] D. Lohmann, and J. Ebert, "A generalization of the hyperspace approach using meta-models", In *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design Workshop,* at 2nd Int'l Conf. on Aspect-Oriented Soft. Dev., 2003. http://www.cs. .bilkent.edu.tr/AOSD-EarlyAspects/Papers/LohEbe.pdf

[23] G. Miller, "Java Modeling: Holonic software development, Part 2", http://www-106.ibm.com/ developerworks/java/library/j-jmod1023/?loc=j#5

[24] Object Management Group. 2001. OMG Unified Modeling Language specification, version 1.4.

[25] M. P. Robillard and G. C. Murphy. "Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies." In *Proceedings of the 24th Int'l Conf. of on Soft. Eng.,* pp. 406—416, 2002.

[26] Shrike, http://dev.eclipse.org/viewcvs/indextech.cgi /org.eclipse.cme/contributions/shrike/

[27] Sun Microsystems, Inc., "J2EE Enterprise JavaBeans Technology", http://java.sun.com/products/ejb/.

[28] S. Sutton Jr. and I. Rouvellou, "Modeling of software concerns in Cosmos", In *1st Int'l Conf. Aspect-Oriented Soft. Dev.,* ACM, 127–133, 2002.

[29] P. Tarr, W. Harrison, and H. Ossher, "Pervasive Query Support in the Concern Manipulation Environment", Submitted.

[30] P. Tarr, H. Ossher, W. Harrison, and S. Sutton Jr., "N degrees of separation: Multi-dimensional separation of concerns", In *21st Int'l Conf. Soft. Eng,* IEEE, , 1999.

[31] B. Tekinerdogan, "ASAAM: aspectual software architecture analysis method", In *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design Workshop,* at 2nd Int'l Conf. on Aspect-Oriented Soft. Dev., 2003. http://www.cs.bilkent.edu.tr/ AOSD-EarlyAspects /Papers/Tekinerdogan.pdf

[32] W. Tracz, "DSSA frequently asked questions", *Software Engineering Notes 19*, 2, 52–56, 1994.

[33] W3C, http://www.w3.org/XML/

[34] D. Wagelaar, "A concept-based approach for early aspect modeling", In *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design Workshop,* at 2nd Int'l Conf. on Aspect-Oriented Soft. Dev., 2004. http://www.cs.bilkent.edu.tr/AOSD-EarlyAspects /Papers/Wagelaar.pdf

[35] E. S. Yu, and J. Mylopoulos, "Understanding 'why' in software process modeling, analysis, and design", In *16th Int'l Conf. Soft. Eng.,* IEEE, 159–168, 1994.