

Early-Stage Concern Modeling

Stanley M. Sutton Jr.

NFA

5 Maple Avenue

Chappaqua, New York 10514 USA

suttonsm@verizonsg.net

ABSTRACT

Concerns are a prominent focus of attention in the early stages of the software life cycle. Particular approaches to requirements engineering and architectural design address particular concerns and relationships, often from a multidimensional perspective. Cosmos is a general-purpose, multidimensional, concern-space modeling schema that can also be used to model early-stage concerns and relationships. Cosmos is not intended to replace other modeling approaches but to supplement and extend them. Cosmos can also be used to bridge the concern gap from early- to later stages of the development life cycle.

1. INTRODUCTION

Aspect-oriented software development (AOSD) attempts to bring the benefits of advanced separation of concerns (ASOC) to the software life cycle. Most work on ASOC to date has focused on the middle, implementation-oriented phases of the life cycle, emphasizing implementation architecture, design, and coding [1,5,4,10,19]. Of course, concerns arise and must be addressed throughout the life cycle, beginning with the early stages of development. Although work in requirements engineering and architectural design has not generally been conducted under the banner of aspect orientation, work in these areas has nevertheless focused quite explicitly on the capture, interrelation, and separation of concerns [20,3,2.12], often with a multidimensional perspective that is consistent with some views of AOSD [19,18].

Given the emphasis on concern modeling and management that can be found in current approaches to requirements engineering and architectural design, it seems that connections to AOSD approaches in later development stages should follow naturally. However, despite the awareness of concerns reflected in these approaches to both early and later stages of development, none of them provides a means for general-purpose concern modeling. Although concerns are heavily modeled, they are modeled mainly in the context of particular tools, methods, or development stages. To capture concerns across the life cycle it is generally necessary to use multiple methods, representations, and models, and the connections between these may be tenuous. Recent work in requirements engineering and architectural design has attempted to span more of the life cycle and provide better connectivity between stages, but the problem is still not addressed in general.

Cosmos has been proposed as a general-purpose concern-space modeling schema [18] that is intended to overcome the limitations of more specialized concern-modeling approaches.

Cosmos provides a framework in which concerns and their interrelationships can be described independently of life cycle stage, development method, and implementation technology.

There are several potential benefits to general-purpose concern-space modeling:

- It can be used to capture concerns in the very initial stages of development, or at other times when they are preliminary or tentative, before they are modeled in requirements, design, or other formalisms
- It can be used to capture concerns that fall outside the scope of other modeling formalisms
- It can be used to relate concerns between different modeling formalisms and development stages
- It can support the modeling and analysis of concerns in the abstract
- It can supplement development tools that may lack a substantial concern-modeling capability

We have used Cosmos independently and with Hyper/J [7] to model the concerns in software components and to support concern-driven component composition [16,17]. Cosmos has also been used for project planning, impact analysis, and development of reusable components [14].

This paper discusses the applicability of Cosmos to the modeling of concerns early in the software life cycle. Section 2 gives an overview of relevant parts of the Cosmos schema, including a running example. Section 3 considers several early-stage modeling approaches, arguing that these reflect considerations appropriate to AOSD and showing with brief examples how they may be addressed in Cosmos. Section 4 provides discussion; Section 5 presents conclusions.

2. COSMOS: A CONCERN-SAPCE MODELING SCHEMA

Cosmos is a general-purpose, multidimensional, concern-space modeling schema. As discussed separately [18], for the sake of generality we consider a *concern* to be any matter of interest in a software system. (This subsumes *aspect* as it has been traditionally defined [9].) We further define a *concern space* as an organized representation of concerns and their relationships.

The Cosmos schema includes three types of elements: concerns, relationships, and predicates. We distinguish between the core schema and extensions to the core schema that may apply to particular installations or applications of the schema.

In this paper we focus mainly on a subset of concerns and relationships, primarily *logical* concerns and *interpretive* relationships (explained below). In describing the schema we give brief examples which are taken from an analysis [16] of a general-purpose software cache, the GPS cache [8]. A goal of this cache is to provide a reusable cache that can be configured

for a variety of applications. The GPS cache has several "top-level" concerns. Some, like core functionality and performance, are typical for caches, whereas others, like generality and management of inter-object dependencies, are more unusual. The Cosmos schema is summarized in Table 1. (Some of the tables and description in this section are abstracted from [18]).

Table 1. Outline of the Cosmos concern-space modeling schema

Core			Extensions
Concerns	Logical	Classifications, Classes, Instances, Properties, Topics	
	Physical	Collections, Instances, Attributes	
Relationships	Categorical	Classification, Generalization, Instantiation, Characterization, Topicality, Membership, Attribution	
	Interpretive	Significance	Admission, Contribution, Logical-Implementation, LogicalComposition, Motivation
	Mapping	Association	Affecting, Description, Modeling, Physical-Implementation, Representation
	Physical	PhysicalRelation	Connection, ConnectionTo, Physically-Affecting, PhysicalComposition
Predicates	<no subtypes defined>		

2.1 Concerns

Cosmos categorizes concerns as *logical* or *physical*. Logical concerns represent the concepts in which we are interested regarding a system or artifact, for example, issues, aspects, features, and properties. Physical concerns represent the system elements or software artifacts to which our logical concerns apply. The two are distinguished because logical concerns can be considered independently of physical concerns and physical concerns can be modeled independently of logical concerns. Both are included because the logical motivate our interests in a system or artifact, whereas the physical allow us to model the system or artifacts in which the logical are realized. Ultimately, it is the relationship between the two on which AOSD depends.

2.1.1 Logical Concerns

Logical concerns represent the conceptual "matters of interest" in a software system. Examples include functionality, behavior, performance, robustness, state, coupling, configurability, usability, size, cost, and so on. Cosmos does not restrict the domain of logical concerns and leaves it to developers (or other stakeholders) to identify concerns of interest. Cosmos distinguishes five types of logical concern: *classifications*, *classes*, *instances*, *properties*, and *topics*.

Classifications represent systems of classes. They are identified with a root class and transitively include the subclasses of that class. From a concern modeling perspective, a classification corresponds to a high-level dimension in a concern space. Because concern-spaces are multidimensional, a particular instance may be concurrently classified according to multiple classifications.

In early work [16?], we categorized concerns in the GPS cache as internal versus external and functional versus non-functional.

External concerns are those seen by users of the system, internal are of concern mainly to developers. Functional concerns include operations and behaviors performed by the system, non-functional include properties and state. All identified concerns were categorized according to both of these dimensions. Subsequently, we identified further classifications, for example, classifications of functionality, of properties, of behaviors, of subsets of behaviors (such as configurability or logging behaviors), and so on. Each of these represents a system of classes of particular types of concern.

Some classifications are independent, such as functional versus non-functional and internal versus external. Others are dependent, such as subclassifications of behavior, which apply to concerns that are already classified as behaviors.

Classifications are important for organizing the organization of concerns. The ability to explicitly discuss classifications makes it possible to explicitly address alternative approaches to the organization and use of concern classes. This is especially important in a multidimensional context, where multiple classifications (and classes) can apply to a given concern and thus where multiple alternative organizations, associations, view points, and access paths for those concerns are possible.

Classes are concerns that are introduced to categorize other concerns. Classes can include other classes (subclasses) and can be used to classify logical instances, properties, and the various kind of physical concern (all discussed below).¹ Classes may be assigned to a classification or may be used apart from any classification.

¹ An implementation of the Cosmos schema in Java is under development that allows classes to apply more generally to all kinds of schema elements.

In the GPS cache, some of the classes of functionality identified include core functionality, functionality related to object invalidation, functionality related to inter-object dependency management, and functionality related to cache configuration. Core functionality (to take one example) can be classified into functions to add objects, delete objects, update objects, and so on. Behaviors can be classified into behaviors to implement functions and other behaviors. Behaviors to implement functions can be classified in parallel with the functions implemented. They can also be classified in other ways, for example, according to performance or concurrency characteristics. Other behaviors include subclasses for logging, buffering, and garbage collection, among others. Alternatively, these can be classified according to whether they may be associated with operations or not and, if so, according to the operations with which they may be associated. (Logging, for example, does not implement any operation but is nevertheless associated with operations, whereas garbage collection neither implements nor is associated with any particular operations.)

Instances are specific concerns that do not classify or characterize other concerns. Instances may be assigned to classes or used independently of any class.

Concern instances in the GPS cache represent specific functions, behaviors, state elements, parameters, and so on. These include, for example, functions to add an object, update an object, and add or update an object; specific behaviors such as those involved in supporting specific functions, in logging operations and statistics, in invalidating out-of-date objects, and in maintaining inter-object dependencies; specific state elements including cache state, object meta-data, and dependency information; and specific parameters such as total cache size and maximum object size.

Properties are concerns that characterize other (logical) concerns. They can be applied to classifications, classes, and instances. Properties applied to classifications apply transitively to classes in the classifications; properties applied to classes apply transitively to members of the class.

Properties are very important in the GPS cache. The primary goal of the cache is to be general. Toward this end, the cache is highly configurable and richly functional. Of course, performance, persistence, concurrency, and recoverability are concerns for many potential applications of the cache, as are correctness of operation and consistency of cached data. The transparency of aspects of the cache implementation is also a concern for its potential impact on users.

Topics are arbitrary collections of concerns. Topics capture theme-related concerns that may belong to several categories. Topics thus provide a way to organize groups of concerns that cut across other Cosmos categorizations. Whereas classifications contain classes of a particular type, and classes contain (sub)classes and members of a particular type, topics may contain elements of any type.

Configurability is a topic we consider important for the GPS cache. This topic includes classes related to configurability (such as configurability functions and configurable behaviors), instances of these classes (specific functions and behaviors, and configurability-related properties). Another topic important for the GPS cache is algorithms, for example, in the areas of efficient dependency management and garbage collection.

2.1.2 Physical Concerns

Physical concerns represent “real world” elements of a system, potentially including software, hardware, systems, and services. As noted above, physical concerns are represented in Cosmos for two main reasons. First, these are the things, such as software artifacts, to which our (logical) concerns apply and through which our logical concerns are realized. Second, these real-world things are of direct concern themselves, not just as derivative or supportive of logical concerns, but as work products, deliverables, and so on.

Cosmos distinguishes three types of physical concern: *instances*, *collections*, and *attributes*. Physical instances represent particular software units, such as .java and .class files. Collections are collections of instances and other (sub)collections, such as packages of code. Attributes are the characteristics of physical instances or collections. (The difference between a physical attribute and a logical property is that the former is a specific characteristic of a specific thing whereas the latter is an abstract concept potentially applicable to many things. For example, performance is a property of interest for the GPS cache, but this general concept is not the same as the specific performance characteristics of specific implementations of the cache.) As physical concerns are not relevant to the examples presented in this paper, they are not discussed further.

2.1.3 Relationships

Cosmos defines four categories of relationship: *categorical*, *interpretive*, *mapping*, and *physical*. The Cosmos core schema defines specific types of categorical relationship, while specific types of the other relationships must be defined according to particular installations or applications.

2.1.4 Categorical Relationships

Categorical relationships relate concerns based on their categories (Table 1). These relationships reflect the semantics of concern types. There are seven types of categorical relationship: *Classification* relates a class to a system of classification. *Generalization* relates classes in an inheritance relationship. *Instantiation* relates instances to classes to which they belong. *Characterization* relates properties to the classes or instances to which they apply. *Membership* relates physical instances to collections to which they belong. *Attribution* relates attributes to physical instances or collections. *Topicality* relates concerns of any type to a topic. These relationships are summarized in Table 2.

Table 2. Cosmos categorical relationships

Element Kind (role)	Categorical Relationship	Element Kind (role)
Classification	Classification	Class
Class (superclass)	Generalization	Class (subclass)
Class	Instantiation	Logical Instance, Property, Collection, Physical Instance, Attribute (member)
Property	Characterization	Classification, Class, Instance
Topic	Topicality	Concern (subject)
Collection	Membership	Physical Instance (member)
Attribute	Attribution	Collection, Physical Instance

2.1.5 Interpretive Relationships

Interpretive relationships reflect interpreted semantic associations among logical concerns. They depend primarily on the context-dependent concern semantics and significance. The Cosmos core schema does not predefine particular interpretive relationship types; these should be added according to concern-modeling needs. In analyzing concerns in the design of the GPS cache, we considered four interpretive relationship types. As indicated in Section 3, relationships such as these are also especially important in early-stage cycle concern modeling. (Indeed, some of these were initially inspired by comparable relationships proposed for requirements engineering.)

Contribution: One logical concern contributes-to another if the way in which, or the extent to which, one concern is addressed affects the way in which, or the extent to which, the other concern is addressed. For example, *optimization* contributes to *performance*. Contribution is not necessarily positive; for example *logging* contributes to *performance* but in a negative way. Contribution is especially important for impact analysis and change propagation. Examples of contribution for concerns in the GPS cache are shown in Table 3.

Motivation: One logical concern motivates another if one concern provides an impetus or justification for the other. For example, *performance* may motivate *configurability* and *recoverability* may motivate *logging*. (Whether these relationships actually hold depends on the particular case.) Motivation supports rationale capture and impact analysis. Some examples of motivation relationships for concerns in the GPS cache are shown in Table 4.

Logical implementation: One logical concern logically implements another if one concern is introduced relative to the implementation of the other. In the GPS cache, a *dependency graph* is introduced to for purposes of implementing *object dependency management*. This is not the same as physical implementation, which would reflect, for example, that a particular code unit implements a particular function.

Table 3. Some examples of contribution relationships for concerns in the GPS cache

Domain	Relationship	Range
Configurability	Contributes-to	Generality
Configurability	Contributes-to (variably!)	Performance
Optimization	Contributes-to	Performance
Logging behavior	Contributes-to (negatively!)	Performance
Logging behavior	Contributes-to	Robustness
Storage optimization	Contributes-to	Optimization
Garbage collection	Contributes-to	Storage optimization
Logging optionality	Contributes-to	Configurability

Table 4. Some examples of motivation relationships for concerns in the GPS cache

Domain	Relationship	Range
Generality	Motivates	Configurability
Performance	Motivates	Configurability
Performance	Motivates (negatively!)	Logging
Recoverability	Motivates	Logging
Performance	Motivates	Optimization
Information hiding	Motivates	Retrieve object-copy function
Performance	Motivates	Retrieve object-original function

Admission: One concern admits another if one concern makes it sensible to consider another. The introduction of *logging* admits *log buffering*. Logging neither requires nor motivates log buffering, but without logging it is meaningless to consider log buffering. Admission reflects the fact that the introduction of some concerns opens the concern space to the consideration of other, semantically dependent concerns.

Some examples of logical implementation and admission relationships for the GPS cache are shown in Table 5.

Table 5. Some examples of admission and logical implementation relationships for concerns in the GPS cache

Domain	Relationship	Range
Logging	Admits	Optionality
Garbage collection AND basic functionality	Admits	Interaction of garbage collection and basic functionality
Cache buffering	Admits	Cache storage allocation
Object dependency graph	Logically-implements	Object dependency management
Object invalidation behavior	Logically-implements	Object dependency management

The interpretive relationship types described above serve particular purposes that were relevant to our objectives in modeling concerns for the GPS cache; others may be defined for other purposes. For example, various kinds of dependencies may be defined for rationale capture among concerns [20].

2.1.6 Physical Relationships

Physical relationships relate physical concerns, for example, reflecting the composition of components into an application. As with interpretive relationships, the core Cosmos schema does not define particular types of physical relationship but allows them to be added according to modeling or analysis needs. The relevance of particular relationships may depend on the kind of physical concerns related, the purpose of the relationship, and on the particular technologies (e.g., development tools) used in working with the physical elements.

2.1.7 Mapping Relationships

Mapping relationships relate logical and physical concerns. *Physically-implements* is an example of a mapping relationship where a physical concern (e.g., a code unit) implements a logical concern (e.g., a function). As with interpretive and physical relationships, the core Cosmos schema defines no particular types of mapping relationships but anticipates that particular types will be defined for particular purposes. Mapping relationships are especially important where logical concerns are used to organize, select, or compose physical concerns (i.e., their corresponding software units) [17].

2.2 Multidimensional Modeling

Cosmos is based on the “multidimensional separation of concerns” premise that concerns in software apply simultaneously in multiple, crosscutting dimensions [19]. Multidimensionality is reflected in several ways in Cosmos. Cosmos defines several kinds of concern, and topics are introduced specifically to capture concerns that cut across the other Cosmos categories. Additionally, Cosmos classifications and classes allow multiple categorizations of concerns. A concern class may be refined according to multiple alternative classifications, a class may be a subclass of multiple super classes, and an instance may belong to multiple classes. Multidimensional concern modeling has proven useful in design and implementation [16,17]; the following examples show that it also applies in early-stage development.

3. EXAMPLES

This section discusses examples modeling approaches from early in the software life cycle. The modeling and interrelation of concerns is already a prominent consideration in these approaches, and many of the notions these approaches incorporate can be readily modeled in Cosmos.

3.1 Preliminary Capture of Concerns

In [16] we describe concerns in the design of the GPS cache. Although this description was formulated after the cache had been implemented, it was based on informal and relatively brief description and discussion of the concerns and their relationships. Thus, in effect this provides an example of the modeling of concerns before they are otherwise formally represented and analyzed, as would occur at the very initial

stages of development. Even in this informal context, we were able to identify a wide variety of concerns and significant semantic relationships among them. As demonstrated by the example in Section 2, all of the kinds of concern recognized by Cosmos came into play, as did several kinds of interpretive relationships (examples in Tables 3-5).

3.2 Requirements Engineering: I* & Tropos

I* is a framework for modeling organizations in terms of actors, goals, and dependencies. Tropos is a methodology that applies this to early requirements and provides a basis for extending early requirements to late requirements, architectural design, and detailed design [20,3].

With respect to concern modeling in requirements, Tropos emphasizes the need to identify organizational concerns, separate these from implementation concerns, and give them first-class treatment. Toward this end, Tropos posits five main classes of concern: actors, resources, (hard) goals, soft goals, and tasks. A particular requirements model will contain multiple instances of each of these classes. Properties are not represented directly in the Tropos schema but may be captured in hard or soft goals (e.g., “increase friendliness of customer service”). Neither are topics (in the Cosmos sense) modeled explicitly, but clearly topics could be useful to modelers, for example, in grouping various instances of the different concern classes according to various application themes (such as all of the goals, agents, and tasks related to customer service).

Tropos (and I*) incorporate several relationships of types that Cosmos would consider interpretive (that is, relating logical concerns according to application-dependent considerations). These include decomposition relationships, means-ends relationships (analogous to Cosmos logical implementation), and dependency relationships (analogous to Cosmos contribution, although Tropos dependencies are ternary whereas Cosmos contribution is binary). Dependencies also relate to the Cosmos notion of motivation.

3.3 Requirements Engineering: KAOS

KAOS [2], like I*, offers a goal-dependency model of requirements and takes a view of concerns that is appropriate to requirements and separated from implementation. Also like Tropos, KAOS provides some downstream continuity, from requirements to architectural refinement.

In some contrast to Tropos, KAOS adopts a more explicitly multidimensional perspective on requirements engineering. This shows up in several ways. Conceptually, requirements in the abstract and elements in a model can be associated with different “aspects” (in their terms) such as “why”, “who”, “when” and “what.” Requirements also have a dual linguistic dimension, including both an “outer” semantic net for general types and semantic relationships and an “inner” assertion language for detailed temporal and logical semantics. Goals can be further classified according to their domain, for example, robustness, safety, efficiency, and privacy. Goals are also subject to disjunctive and conjunctive refinement, with the former providing alternative realizations in an implementation. The KAOS system also supports multiple views of the requirements, including refinement, operationalization, entity-relationship, and agent.

These alternative dimensions of requirements can be represented as classifications in Cosmos. Each such concern classification may have a number of concern classes, such as classes of language (semantic net versus assertions), classes of linguistic constructs (such as agents, goals, constraints, temporal assertions), classes of goals, and so on. As in Tropos, properties per se are not a first class construct in KAOS but are naturally represented by other constructs such as goals and constraints. Neither are topics (in the Cosmos sense) first class constructs, but as with Tropos these could be quite useful with a KAOS model. (To a certain extent, the KAOS views represent the instantiation of particular types of topics.)

As noted, relationships in KAOS include conjunctive and disjunctive goal refinement, which relate to the Cosmos notions of contribution, motivation, and logical implementation. KAOS relationships also include operationalization and responsibility, which reflect further aspects of contribution and motivation.

3.4 Architectural Design: ABAS

ABAS are “attribute-based architectural styles” [11]. These architectural styles are designed to address specific quality attributes and they can be analyzed in terms of these attributes.

ABAS modeling is explicitly multidimensional. It incorporates a number of classifications, including classification of architectural attribute information (in terms of external stimuli, architectural decisions, and responses), classification of architectural elements (in terms of components, connectors, and properties), classification of ABAS specification elements (such as problem description and stimulus/response attribute measures), and classification by property. Additionally, parameters in each of the categories of information are subject to multiple simultaneous classifications. For example, stimuli are classified with respect to mode, source and regularity, and responses are classified with respect to latency, throughput, and precedence. (These are just two of many examples.)

ABAS, in contrast to the requirements methods described, treats properties explicitly and prominently. All architectural styles are classified with respect to the properties of performance, modifiability, and availability. These crosscut the information categories, so that the kinds of information used to describe stimuli, architectural decisions, and responses for performance are different from those used for modifiability. For example, performance responses can be described in terms of latency and throughput, modifiability responses in terms of extent of impact and effort of change.

ABAS does not include the Cosmos topic notion, but (like views in KAOS) certain classifications in ABAS (such as the style specification elements) can be considered specialized instantiations of topics (in that they pull together concerns from various other dimensions).

The sorts of relationships that are emphasized in the requirements modeling approaches (dependency, responsibility) and in Cosmos category of interpretive relationships are not highlighted in ABAS. Of course, ABAS describe kinds of physical relationships (connections) among components that observe various architectural styles. Additionally, ABAS include potentially detailed and quantitative analytical models that describe how specific property measures relate to architectural elements and changes to those elements. Based on

these models it should be possible to derive higher-level associations reflecting dependency, implementation, contribution, and so on.

3.5 Business Architecture

The previous approaches address early concern modeling for application development. Concern modeling is also relevant in other sorts of modeling and development. I have conducted exercises with Cosmos in two other domains, business architecture and development methodology. In both of these cases, Cosmos concepts were generally successful for describing key aspects of the modeled domain.

A business-architecture model was taken from [13]. The described architecture can be modeled in terms of classifications (and classes) including, among others, architecture domains (e.g., business and technology), development paths (e.g., based on existing assets or enterprise knowledge), and business concepts (e.g., business purpose, business situation, and business outcome). The classifications are multidimensional in that certain elements may be classified in multiple ways (e.g., a business role-player may be classified by location, by type, and by capability). Specific instances of these concern classes, of properties, and of topics can be introduced in designing the architecture of specific enterprises. For instance, “customers retained” may be a specific instance of a business outcome, “user-friendliness” may be a property of an automated customer inquiry system, and “customer relations” may be a topic that gathers a number of different properties and concerns from different classes.

The business architecture also includes 19 kinds of relationships that Cosmos would mainly classify as interpretive. Many of these correspond closely to defined Cosmos relationships. Motivation is represented by “motivates”, admission is represented by “defines”, various kinds of contribution are represented by “alters”, “creates”, “governs”, “produces”, and “supports”, and aspects of implementation are represented by “assigned-as”, “enables”, “fulfills”, “performs.” A few of the relationships would be extensions for Cosmos, such as “binds”, “consumes”, and “manipulates.”

The significance of business architecture modeling in Cosmos, as with organizational modeling in I* [20], is that it shows that concerns representing users and the (enterprise) operating environment can be modeled analogously to concerns in the software itself. That in turn suggests that the two may be more effectively integrated both in principle and in practice.

4. DISCUSSION

Concerns are plainly a prominent focus of attention in the early stages of development. Requirements specification and analysis aim precisely to introduce concerns into the life cycle from the user or enterprise perspective; architectural design begins to raise and address concerns related to implementation.

Approaches in both requirements and architecture adopt a multidimensional view of concerns. This is reflected, within individual approaches, in the use of multiple conceptual perspectives, multiple languages, multiple views or specifications, and multiple classifications. Additionally, a multidimensional view of requirements engineering overall has been emphasized by van Lamsweerde [12]. He characterizes the first 25 years of requirements engineering research as

addressing the four dimensions of ontology, structuring, specification, and reasoning, and he argues that the next 25 years will focus on “multi-X” approaches, integrating multiple languages and formats, multiple facets and viewpoints, and multiple modes of reasoning.

Cosmos seems substantially able to represent the kinds of concerns and relationships proposed for early-stage modeling.

- Modern requirements engineering approaches propose specific kinds of concerns (such as goal, agent, etc.). Cosmos does not directly include specific concern kinds like these, but they can be defined using the ontological mechanisms in Cosmos (classifications, classes, and instances). Cosmos is thus on a meta-level with respect to specific concern kinds, analogous to the relationship between UML [15] and specific object kinds (classes).
- Cosmos recognizes properties as an explicit category of concern, which the requirements engineering methods do not. However, these methods do support the modeling of properties in other guises (such as goals), and properties are commonly addressed in specific models. Additionally, properties are a first-class notion in ABAS. Thus, the incorporation of properties into the Cosmos schema seems well justified for early-stage concern modeling.
- The requirements and architectural approaches do not incorporate an explicit notion analogous to Cosmos topics, which support arbitrary grouping and indexing of concerns. These particular approaches may be complete enough in themselves that generalized topics are not needed (although particular kinds of viewpoints and specifications in some of these approaches may be seen as instantiations of specific kinds of topics). Nevertheless, across a broader range of life cycle activities and artifacts, and possibly as a supplement to early-stage approaches, the general notion of topics should still be useful.
- Regarding the interpretive relationships (generally based on interpretation of application or problem semantics), the specific relationships we have previously defined for Cosmos seem to address more or less the same issues as relationships proposed in requirements modeling. However, in particular approaches the particular relationships (e.g., forms of dependency) often vary. Cosmos is intended to be extensible with respect to such relationships so as to accommodate specializations for specific approaches. What is most important for the purposes of Cosmos is that relationships of these types be addressed; the particular varieties of relationship should be tailorable as needed.

Based on the experience with early-stage concern modeling, combined with experience in concern modeling for design and implementation [16,17], it seems that general-purpose concern modeling can address concerns across a significant range of the life cycle, larger than those spanned so far by particular early- (and later-) stage modeling approaches. Additionally, it seems that general-purpose concern modeling may extend and

supplement the kinds of early-stage concern modeling that are now performed, supporting concern modeling in different contexts or for different purposes, and possibly enabling different perspectives on concerns to be captured. However, it is not the intention of Cosmos to replace traditional forms of concern modeling or documentation but to be used with them.

Some of the benefits and applications of Cosmos are due to its relatively general nature compared to specific modeling approaches. Would a scheme that was more general still be even more useful? Approaches such as relational and entity-relational do not embody concepts of concern modeling as first class elements. Cosmos is able to leverage the specialized semantics of that domain. Object-modeling approaches, such as UML [15], have some but not all of the concepts embodied in Cosmos. Their generic mechanisms can be used to represent concern-modeling notions, but not as first class elements. Also, the domain of object-modeling is not the same as that of concern modeling, as concerns are mainly concepts, not objects.

5. CONCLUSIONS

Concerns are a prominent focus of attention in the early stages of the software life cycle, including both requirements engineering and architectural design. Particular approaches in these areas address particular concerns and relationships, often from a multidimensional perspective.

Cosmos is a general-purpose concern-space modeling schema. It represents both logical concerns (“matters of interest”) and physical concerns (systems elements). Logical concerns are categorized in terms of classifications, classes, instances, properties, and topics. Kinds of relationship in Cosmos include categorical, interpretive, mapping, and physical. This general modeling schema is substantially able to represent early stage concerns and to capture or approximate important relationships among these concerns. Cosmos has previously been used to model concerns in the design and implementation stages.

Cosmos is not intended to replace other forms of modeling or documentation but to be used with them. Potential applications include preliminary concern modeling (prior to more formal or specialized modeling), supplemental concern modeling (addressing concerns or perspectives not otherwise covered), basic concern modeling for tools that may lack the capability, the linking of concerns across stages, artifacts, and approaches, and indexing into more comprehensive models and documents. Through such applications Cosmos can enrich the modeling of early-stage concerns and help to strengthen the connections between early- and later-stage concerns and between the activities and artifacts in which these are addressed.

6. ACKNOWLEDGMENTS

For earlier collaboration on Cosmos I thank Isabelle Rouvellou. For discussion on MDSOC and Hyper/J I thank Peri Tarr and Harold Ossher. For sharing his experience with Cosmos I thank Juri Memmert. For help with the GPS cache I thank Arun Iyengar and Lou Degenaro. For additional discussions on MDSOC I thank Stefan Tai and Thomas Mikalsen.

7. REFERENCES

- [1] Aksit, M. Wakita, K. Bosch, J., Bergmans, L., and Yonezawa, A. Abstracting object-interactions using composition-filters. In R. Guerraoui, O. Nierstrasz, and M. Riveill, Eds., *Object-based Distributed Processing*, Springer, Verlag, 1993.
- [2] Bertrand, P., Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A. GRAIL/KAOS: An Environment for Goal Driven Requirements Engineering 20th 22nd Int. Conf. on Software Eng. IEEE-ACM, Kyoto, April 1998.
- [3] Castro, J., Kolp M., and Mylopoulos, J. Towards Requirements-Driven Information Systems Engineering: The Tropos Project, 35 pages. To appear in *Information Systems*, Elsevier, Amsterdam, The Netherlands, 2002.
- [4] Clarke, S., Harrison, W., Ossher, H., and Tarr, P. Towards Improved Alignment of Requirements, Design, and Code. Conf. on Object-Oriented Programming, Systems, Languages, and Applications, Denver, Colorado. ACM SIGPLAN Notices, v. 34, n. 10, pp. 325--339, 1999.
- [5] Harrison, W. and Ossher, H. Subject-oriented Programming (a Critique of Pure Objects). Conf. on Object Oriented Programming: Systems, Languages, and Applications, Sep. 1993.
- [6] Harrison, W., Ossher, H., and Tarr, P. Software Engineering Tools and Environments: A Roadmap. In Finkelstein, A., ed., *The Future of Software Eng.*, 22nd Int. Conf. on Software Eng., Limerick, Ireland; ACM, New York, pp. 263-277. June, 2000.
- [7] IBM. HyperJ. <http://www.research.ibm.com/hyperspace/HyperJ/>.
- [8] Iyengar, A. Design and Performance of a General Purpose Software Cache. In Proc. of the 18th IEEE Int. Performance, Computing, and Communications Conf. (IPCCC'99), Phoenix/Scottsdale, Arizona, Feb. 1999.
- [9] Kiczales, G., Lamping, J. Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., and Irwin, J. Aspect-Oriented Programming. European Conf. on Object Oriented Programming, Finland. Springer-Verlag LNCS 1241, June 1997.
- [10] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. Getting Started with AspectJ. Comm. of the ACM, v. 44, n. 10, pp. 59-65. Oct. 2001.
- [11] Klein, M. and Kazman, R. Attribute-Based Architectural Styles. Technical Report CMU/SEI-99-TR-022, Oct. 1999. (<http://www.sei.cmu.edu/publications/documents/-99.reports/99tr022/99tr022abstract.html>)
- [12] van Lamsweerde, A. Requirements Engineering in the Year 00: A Research Perspective. 22nd Int. Conf. on Software Eng., ACM Press, 2000, pp. 5--19.
- [13] McDavid, D. A Standard for Business Architecture Description", IBM Systems Journal, v. 38, n. 1, pp. 12--31, 1999.
- [14] Memmert, J. Employing AOSD Technologies in Large Companies. 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April, 2002 (to appear).
- [15] Object Management Group. OMG Unified Modeling Language Specification, version 1.4, Sep. 2001.
- [16] Sutton Jr., S. M. and Rouvellou, I. Concerns in the Design of a Software Cache. Workshop on Advanced Separation of Concerns in Object-Oriented Systems. Conf. on Object-Oriented Programming, Systems, Languages, and Applications, Minneapolis, Minnesota, Nov. 2000.
- [17] Sutton Jr., S. M. and Rouvellou, I. Advanced Separation of Concerns for Component Evolution. Workshop on Engineering Complex Object Oriented Systems for Evolution. Conf. on Object-Oriented Programming, Systems, Languages, and Applications, Tampa, Florida, Oct. 2001.
- [18] Sutton Jr., S. M. and Rouvellou, I. Modeling of Software Concerns in Cosmos. 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April, 2002 (to appear).
- [19] Tarr, P., Ossher, H., Harrison, W. and Sutton Jr., S. M. N Degrees of Separation: Multidimensional Separation of Concerns. 21st Int. Conf. on Software Eng.. ACM, New York, 1999, pp. 107--119.
- [20] Yu, E. S. and Mylopoulos, J. Understanding "Why" in Software Process Modeling, Analysis, and Design. 16th Int. Conf. on Software Eng. (ICSE 16), Sorrento, Italy; IEEE, 1994, pp. 159—168.