

# Issues in the Design and Implementation of a Concern-Space Modeling Schema

**Stanley M. Sutton Jr. and Isabelle Rouvellou**

IBM T. J. Watson Research Center  
30 Saw Mill River Road  
Hawthorne, New York 10532, USA  
{suttonsm, rouvellou}@us.ibm.com

## Introduction

In this paper we address issues in the design and implementation of a concern-space modeling schema. A concern space is a representation of the concerns that apply to a software system and of various relationships among those concerns. This work is based on the premise that concerns should be treated as first class entities in software development. Concern-space modeling is key to enabling concerns to be treated independently and systematic across the software life cycle.

We are developing a prototype concern-space modeling schema called Cosmos, in conjunction with which we are also experimenting with concern modeling for middleware and other systems [6]. The design and implementation of Cosmos entails the definition of a data model for concerns and their relationships and the design and implementation of structures to maintain data according to that model.

Concern-space modeling is a new domain, and Cosmos is a prototype, so the issues we face are primarily issues in understanding the concern-space modeling domain and applications in that domain. Our initial objectives in developing Cosmos are thus primarily to gain an understanding of the domain and its applications. Below we first review our perspective on concerns and concern-space modeling, then describe some of the more important issues we face at this time.

## Background

We give our definition of a concern, describe our motivation for modeling concerns, review our requirements for a modeling schema, and indicate some applications for such a schema.

### What is a Concern?

Every good software engineer intuitively knows what a concern is and should naturally strive to keep concerns separate during development. Yet, despite the importance of separation of concerns in software engineering, there have been few attempts to define *concern*. Recent definitions have tended to identify concerns with software artifacts [8,3]. This is natural, since much previous work on separation of concerns has been focused on the organization or construction of software artifacts according to concerns. However, an identification of concerns with artifacts seems restrictive, for two reasons. First, it is possible to consider concerns in a software system independently of artifacts. Second, concerns must be considered with respect to many kinds of artifacts, as new concerns arise at each stage of development, and many concerns cut across development stages and their corresponding artifact types. A definition of *concern* that is tied to artifacts also seems non-intuitive.

In contrast, one dictionary definition of *concern* is “matters for consideration.”<sup>1</sup> This definition seems not only natural, but also appropriate for many software engineering purposes. It is conceptual, that is, not tied to artifacts, but it does not restrict the kinds of artifacts to which concerns can be related. Additionally, it also does not restrict the domain of concerns or their relationships. For these reasons, we generally consider a concern to be any matter of interest in a software system.

---

<sup>1</sup> Merriam-Webster Online Dictionary, <http://www.m-w.com/cgi-bin/dictionary>.

## Why Model Concern Spaces?

The satisfaction of concerns can be considered the essential purpose of software, and separation of concerns is one of the hallowed principles of software design. Moreover, concerns arise at every stage of the software life cycle and are reflected in every software artifact. More particularly, we can envision many significant applications of a concern-space modeling schema (and of particular concern-space models). For example:

- Capturing and abstracting concern-related information from successive stages of a development process and propagating that information to subsequent stages (more generally, acting as a semantic coordination framework across the life cycle)
- Assessing the impact of changes, such as adding a feature or removing a function, in terms of the concerns addressed, eliminated, or otherwise affected
- Propagating changes, according to concerns, across a range of associated life-cycle artifacts
- Evaluating system components for possible reuse: What desirable concerns do they address? What undesirable concerns do they include? How “modular” or “cohesive” are the concerns they embody?
- Managing product families in which members are specified and organized by concerns
- Generating customized applications that are tailored in terms of concerns
- Providing a basis for concern-oriented systems integration techniques
- Guiding development activities in general according to particular concerns raised by the processes, approaches, or components used (e.g., guiding the design in a message-based architectures according to identified “messaging” concerns)

## Requirements on a Concern-Space Modeling Schema

Based on the desire to support applications such as those listed above, we identified particular requirements for a concern-space modeling schema in [7]. Our overarching concerns in these requirements are generality and independence. Concern-space modeling is itself a crosscutting concern in software development: a general-purpose concern-space schema should cut across life-cycle stages, artifact types, and development methods. It should allow for the capture and interrelation of concerns appropriate to a variety of methods, projects, and products, and it should enable concerns to be associated to, and used to organize, a variety of artifact types and system components.

## Issues

The issues we are investigating in the Cosmos design and implementation generally derive from two sources. One is the general nature of concerns and concern spaces. The other is the nature of applications, such as those described above, that are based on concern-space schemas and models. Many issues reflect considerations in both areas.

### Generality of Representation

One of our main goals is to create a concern-space model that is independent of particular life-cycle technologies, methods, stages, and artifacts. We have designed a schema for the model that is not based on the representation of concerns or concern relationships that is used in any particular concern-based development tool. However, we also hope that our schema can be used with a variety of specific concern-based technologies. We will investigate the extent to which integration with particular methods or tools may complicate or compromise the generality of a concern-space schema. So far we have found that the general sort of schema we have adopted for Cosmos has worked fairly well to capture Hyper/J [1] integration relationships, which describe how units of Java code are to be composed (a few more comments on this are given below).

### Role with respect to Documentation

A concern-space modeling schema is, *de facto*, a kind of documentation template in that it represents kinds of information about concerns and their interrelationships. Such information may be put directly, *ab initio*, into a concern-space model, but it will generally be derived from other sources, namely the artifacts ordinarily produced during software development. We do not intend that a concern-space model should replace or duplicate other documentation. Rather, it should abstract and synthesize concern-descriptive information from various sources while referencing those sources for information on the topics indicated by the concerns. Thus, we see a concern-space schema as providing a kind of semantic hyper-index into a collection of software artifacts or system components.

## Typing of Concerns

Concerns can be classified in many different ways. In [6] we distinguished functional versus nonfunctional and external versus internal concerns in the design of a general-purpose cache, the GPS cache [2]. Other classifications could have been used. However, it is not clear that such classifications represent fundamental types of concerns so much as attributes on or views of concerns. In Cosmos, we have so far focused on a few simple and general considerations in the typing of concerns.

**Logical versus Physical Concerns** The first distinction we make is a semantic one between logical and physical concerns. Logical concerns represent the notion of a concern as a “matter for consideration.” They are abstract, conceptual entities that are independent of any embodiment in artifacts. By acknowledging logical concerns directly we help to raise concerns to first-class status. Physical concerns, in contrast, represent the artifacts of a software system, including actual software units, hardware units, systems, and services.<sup>2</sup> There are two reasons for including physical concerns. First, artifacts and other concrete entities are also “matters for consideration”, if not the only ones. Second, physical concerns provide hooks for relating conceptual entities to actual software and systems.

**Simple Concerns versus Concern Groups** The second distinction we make is a structural one, that is between simple (or individual) concerns and composite (or grouped) concerns. (“Simple” in this case refers to structural rather than semantic simplicity.) There is both a semantic and a structural reason to include concerns that are groups of concerns. The semantic reason is that some concerns emerge as a combination of other concerns, for example concerns that arise from the interaction of features. The structural reason is that it facilitates the representation of many-to-many relationships among concerns.

**Subtypes of Logical and Physical Concerns** A further issue is whether subtypes of logical and physical concerns can or should be distinguished. Logical concerns, although they represent many different specific concepts, all seem to be similarly conceptual in essence. Additionally, logical concerns generally seem to be subject to the same sorts of conceptual relationships (discussed below). Although categories of logical concern may be recognized, these do not seem inherent to the concerns. For these reasons, we have not distinguished subtypes among logical concerns. It will be interesting to see if further work in concern modeling provides a motivation to do so.

Physical concerns, in contrast, or the entities they represent, are usually considered to be typed: software versus hardware, code versus design, different types of code or design, and so on. Additionally, the kinds of relationships that may exist among physical concerns, and possibly between physical concerns and logical concerns, may reflect this typing. For these reasons, it seems natural and potentially useful to be able to represent subtypes of physical concerns. An issue in this case, though, is how much of the potentially extensive typing information for physical concerns will be needed by particular applications of a concern-space model. A concern-space modeling schema intended to support a comprehensive, full life-cycle IDE may benefit from a richly subtyped model of physical concerns. However, particular applications of a concern schema are more likely to be focused on a relatively limited subset of physical concern types. Rather than trying to impose a comprehensive type model of physical concerns at this time, we have not elaborated any physical concern subtypes in Cosmos. However, the need to work with different sets of physical concerns in different applications argues for an extensible model of physical concern types.

**Attributes and Properties of Concerns** Another issue in the typing of concerns is concern attributes and properties. Logical concerns all require some form of identification and description. They may be further characterized by a common set of attributes, such as priority, or properties, such as well-formedness criteria. So far, we have not discovered any particular attributes or properties that would generally apply to one set of logical concerns and not another. Of course, the subjects of various concerns might be characterized by very different kinds of information for various purposes. A requirement for performance would be expressed very differently from a requirement for functionality or for reliability. However, we expect such differences to show up in the artifacts represented by physical concerns to which the logical concerns are mapped, not in the descriptions of the logical concerns themselves. We expect to elaborate the set of attributes and properties for logical concerns as we gain experience with concern modeling.

---

<sup>2</sup> Within the model physical concerns are references to actual pieces of software or hardware

Physical concerns, in contrast again, may be characterized by a variety of attributes and properties, consistent with the variety of types of entities represented. As with subtypes, rather than try to enumerate a comprehensive model, it seems more practical, at least initially, to provide a general or extensible mechanism for defining attributes and properties of physical concerns.

## Typing of Concern Relationships

As with the typing of concerns, there are issues in the typing of concern relationships.

**Kinds of Mapping** We distinguish four types of concern relationship based on the types of concerns related: logical to logical, logical to physical, physical to physical, and physical to logical. We refer to these as “kinds of mapping” to distinguish them from types of relationship that may be based more on relationship semantics. For example, we expect that each kind of mapping will include multiple kinds of semantic relationship among concerns (particular cases of which are described below).

Each mapping kind relates logical or physical concerns in general, regardless of whether they are simple concerns or concern groups. That allows each kind to represent one-to-one, one-to-many, and many-to-many relationships. We have seen no need for a relationship among generic concerns (i.e., regardless of whether they are logical or physical). We have also seen no need for types of relationship that are restricted to simple concerns or concern groups.

Each of our mapping kinds is directional, because we believe that directionality will be implied by most (if not all) of the relationships that most applications will require. Additionally, all of our mapping kinds are binary, relating concerns in two distinguished roles (source and destination). It is possible to construct n-ary relationships from binary relationships, and the ability to relate groups of concerns adds additional modeling flexibility to the mapping kinds, but it remains to be seen whether direct support for n-ary mapping kinds would be useful.

**Subtypes of Relationship** We see a real need to distinguish subtypes of relationship based on relationship semantics. Capturing semantic relationships among concerns is a big part of the value that may be added by a concern-space modeling schema. We expect that generally different semantic subtypes of relationship will be defined for each of the mapping kinds. For example, for physical-to-physical mappings, we can define a set of relationship types corresponding to Hyper/J integration relationships. These relationships, such as “bracket” and “merge by name”, make little sense for the other mapping kinds. In analyzing logical concerns in a software cache, the GPS cache, we identified four important types of semantic relationship, namely *contribution*, *motivation*, *admission*, and *implementation*. A detailed explanation of these is beyond the scope of this paper, but, briefly, *contribution* reflects the impact of one concern on another, *motivation* reflects the motivation of one concern by another, *admission* represents that the introduction of one concern makes possible (or sensible) the introduction of another concern, and *implementation* indicates that one concern is introduced with respect to the implementation of another concern.<sup>3</sup>

We also expect that different applications will require different relationships. For example, we would not expect the Hyper/J integration relationships to be appropriate for aspect-oriented programming [3]. The particular logical relationships we identified for the GPS cache might not be entirely appropriate (or sufficient) for describing the concerns associated with system architectures. Given the apparent application-dependence of types of concern relationship, it seems appropriate to allow for the extension or specialization of concern relationship types. It will be interesting to see whether, as experience is gained, there is convergence toward a generally accepted or widely used sets of relationship types. We hypothesize that this is more likely on the logical than the physical side.

**Attributes and Properties of Relationships** As with concerns, it seems that there are at least a few general attributes, such as name and description, that characterize all types of relationship. Other attributes or properties that may be widely useful include, for example, priorities, weights, and numerical or predicate-based constraints. The representation of some physical relationships seems likely to entail the introduction of type-specific attributes and properties, for example, designations of “optional” versus “required” on a composition relationship, or quality parameters on a service relationship. The need for type-specific attributes or properties on logical relationships

---

<sup>3</sup> In our experience, each of these relationships reveals something interesting and distinctive about concerns in the cache. Unfortunately, a more detailed examination of the relationships must be deferred to another paper.

seems less clear at this time, although it cannot be precluded. More experience needs to be gained before more precise characterizations of the needs for relationship attributes and properties can be made.

### **Kinds of Access**

A concern-space modeling schema may serve various purposes and thus need to be viewed and accessed in a variety of ways. Suppose we are working with a concern-space model of the GPS cache, and we are considering the possibility that we may want to remove the logging behavior of the cache in order to improve performance.<sup>4</sup> We may ask the following questions:

What are the concerns that motivate logging? If I remove logging for the sake of performance will I be leaving other concerns (say, robustness) unaddressed? What are the concerns that logging affects (apart from those that may motivate it)? What are the various code units that support logging? What other concerns do those particular units support or affect? And so on ....

Such questions may be addressed by a combination of query and navigation, each of which may be appropriate under different circumstances. A concern-space model schema should thus be implemented with both of these modes of access in mind.

Abstractly, the Cosmos schema is hyper-multigraph (as we hypothesized in [6]). In other words, we have a graph in which nodes may be sets of nodes, and in which nodes may be connected by multiple edges (of one or more types). As noted above, the edges are directional with respect to their semantics, but they may need to be traversed in either direction. Our initial implementation for Cosmos uses tables to store both concerns and relationships (corresponding to a set-of-nodes-and-edges view of the graph). The tables can be normalized to minimize data redundancy, thereby simplifying the upkeep of stored models. The tables make it fairly easy, if not necessarily efficient, to support both query and (bi-directional) navigational access. Efficiency can be enhanced through the use of indices.

In support of particular visualizations or applications, we are anticipating creating derived views of concern-space models. For example, to support a graphical view (that is, one presenting nodes and edges), we may derive a nodes-and-edges-based representation. To support an outline view, we may derive an outline-based representation. For many purposes, we expect that it will be feasible to “batch process” the derivation of such representations.

### **Consistency and Integrity**

We expect that a well-formed Cosmos model will be subject to a number of integrity constraints. For example, the relationships in a model should be restricted to relating concerns that are in the model, and relationships intended for logical concerns should not reference physical concerns. Other integrity constraints will depend on the semantics of particular relationships. For example, the admissions relationship mentioned above should not be circular, although circularity may be allowed for contribution (since different kinds of contribution may be made in different directions). Additionally, we recognize a number of logical relations among the four kinds of logical-to-logical relationship mentioned above. For example, the set of concerns that motivate a given concern should be a subset of the concerns to which the given concern contributes. As we gain experience and extend the set of concern and relationship types, we expect that our understanding of integrity in concern-space schemas will grow.

We also expect that particular applications, products, and processes will require additional particular consistency conditions. Currently such conditions can only be evaluated and enforced from outside the Cosmos model. As the Cosmos architecture evolves, it may become appropriate to provide mechanisms by which externally defined consistency conditions can be incorporated into Cosmos. The Accessible Business Rules system [4] may be appropriate for this purpose.

Consistent with our prior work on accommodating inconsistency during software development [5] we do not expect that all Cosmos models will be well formed (or otherwise consistent) all the time. Thus, Cosmos will require consistency management mechanisms that allow for flexibility in enforcement.

---

<sup>4</sup> Logging is actually optional in the GPS cache, but ignore that for the sake of the example. Anyway, even optional logging may have some influence on performance.

## Extensibility and Stability

We have highlighted the need for extensibility with respect to several aspects of a concern-space modeling schema. However, we also expect that stability of a schema (and of particular models) will be important for many applications. For example, changes to a schema (or a particular model) may affect the integrity and consistency of a model or otherwise compromise the results of ongoing operations. Thus, there may be a delicate balance between the needs for extensibility and stability of a schema. Moreover, this balance may be different for different development processes, environments, or products. Requirements for extensibility and stability need to be evaluated in different contexts and appropriate extensibility mechanisms and controls designed accordingly.

## Conclusion

Space precludes discussion of further issues, although there are many others (performance, scalability, transactional properties, event integration, reuse, and so on). Our work on and with Cosmos is in the early stages still, but already we have made some interesting observations about concern spaces and concern-space modeling. We advocate a general-purpose, adaptable approach based on a distinction between logical and physical concerns, but many fundamental questions about the nature and use of such concern spaces remain to be investigated.

## References

- [1] IBM. Hyper/J. <http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm>
- [2] Iyengar, A. Design and Performance of a General Purpose Software Cache. In Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference (IPCCC'99), Phoenix/Scottsdale, Arizona, February 1999.
- [3] Kiczales, G., Lamping, J. Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., and Irwin, J. Aspect-Oriented Programming. In Proceedings of the European Conference on Object Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, June, 1997.
- [4] Rouvellou, I., Degenaro, L., Rasmus, K., Ehnesbuske, D. and McKee, B. Extending Business Objects with Business Rules. In Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Europe 2000), IEEE Computer Society Press, pp. 238-249, 2000.
- [5] Sutton Jr., S. M., Heimbigner, D. and Osterweil, L. J. APPL/A: A Language for Software Process Programming. ACM Transactions on Software Engineering and Methodology, v. 4, n. 3, pp. 221-286, July, 1995.
- [6] 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, OOPSLA 2000.
- [7] Sutton Jr., S. M. and Rouvellou, I. Modeling of Concerns and Concern Spaces: Rationale and Requirements. In submission.
- [8] Tarr, P., Ossher, H., Harrison, W. and Sutton Jr., S. M. N Degrees of Separation: Multidimensional Separation of Concerns. In Proceedings of the 21st International Conference on Software Engineering. ACM, New York, 1999, pp. 107--119.