

# Concern Space Modeling in Cosmos

Stanley M. Sutton Jr. and Isabelle Rouvellou

IBM T. J. Watson Research Center

Hawthorne, NY 10532

{suttonsm, rouvellou}@us.ibm.com

## ABSTRACT

Cosmos is a schema for modeling software concerns across the life cycle. It defines a metamodel, including concern types, relationships, and predicates, for modeling multidimensional concern spaces. Cosmos allows concerns to be modeled independently of development formalisms, tools, and methods, and it complements and supports advanced separation of concerns technologies.

## Keywords

Advanced separation of concerns, multidimensional separation of concerns, concern modeling, software engineering

## 1. INTRODUCTION

Concerns are what we care about in software. Software is developed to address concerns expressed in functional and nonfunctional requirements. Software must further reflect and accommodate concerns that arise in the development process.

Separation of concerns is a hallowed principle in software engineering. The goal of separation of concerns is to allow independent concerns to be represented independently in software. Advantages of separation of concerns include the ability to partition development efforts according to concerns and the ability to isolate changes, minimizing modification efforts and limiting the impact on unrelated parts of a system.

Effective separation of concerns should make software easier to maintain, evolve, customize, adapt, adopt, integrate, and reuse. However, such activities remain difficult and costly. As argued in [8], this is due at least in part to an imbalance between the complexity of concerns and the capabilities of current mechanisms for separating concerns. Any given software component is subject to multiple, simultaneous, overlapping concerns, whereas most programming languages and other development formalisms support the systematic separation of only

a small number of concerns (such as classes or functions). As a result, many concerns must crosscut the primary decomposition and become entangled with it. Changes related to crosscutting concerns are consequently not isolated and may entail disproportionately large impacts and costs.

A number of approaches to advanced separation of concerns (including SOP/D [2,1], AOP [5], and Hyperspaces [8]) have been proposed to support greater modularity of concerns and ease of change. These are primarily compositional technologies focused on the creation and manipulation of software artifacts (principally, so far, code and designs). With Cosmos we present a complementary approach to advanced separation of concerns that addresses concern modeling as an independent concern in software development.

## 2. PROBLEM AND APPROACH

The modeling of concerns is a major aspect of modern software development. Nevertheless, concerns are not yet first class citizens of the software life cycle. Concerns are represented differently in different work products and life-cycle stages, and links between different manifestations of concerns in different forms and at different stages are often represented incompletely or implicitly, if at all. There is no global model of concerns, no systematic representation of concerns that is artifact- or stage-independent, and no consistent or comprehensive catalog of concerns. As a result, concerns in different work products or at different stages cannot be handled uniformly, reasoning about concerns in general (across or apart from the life cycle or work products) is inhibited, and concern-oriented tools and methods lack a common foundation.

The goal of concern modeling is to address these limitations of current approaches: to provide a global concern-space model including concerns and their interrelationships, to enable concern spaces to be represented systematically and independently of particular artifacts, formalisms, development tools, and methods, and to allow concerns to be represented consistently and comprehensively. Concern-space modeling should enable concerns to be treated uniformly across the life cycle and range of work products, should facilitate reasoning about concerns in general, and should provide a common foundation for concern-oriented tools and methods.

## 3. THE COSMOS SCHEMA

Cosmos is a concern-space modeling schema. Cosmos represents concern spaces in terms of *concerns*, *relationships*, and *predicates*.

Cosmos divides concerns into two categories, *logical* and *physical*. Logical concerns represent concerns viewed conceptually: topics, issues, properties, problems, in general “matters for consideration” [6]. Physical concerns represent the

ACM COPYRIGHT NOTICE. Copyright © 2001 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

real things of which our systems are comprised, work products, hardware, services, resources, etc. Logical concerns are further typed as *kinds*, *instances*, *properties*, and *topics*. Kinds are groups of concerns of a particular type (such as functions, behaviors, states). Instances are particular concerns of some type (e.g., particular functions, behaviors, states). Properties are characteristics of kinds and instances of concerns. Topics are (typically theme-related) groups of concerns of generally different types or kinds. Physical concerns are typed as *instances* or *collections*. Instances represent particular system elements (such as source files, design documents, workstations); collections represent groups of these.

In a caching system we have modeled [4,7], *functionality*, *behavior*, and *state* are kinds of concerns, *create object* and *toggle logging* are instances of concerns, *performance* and *robustness* are properties, and *logging* and *algorithms* are topics.

Relationships are divided into four categories: *categorical*, *interpretive*, *physical*, and *mapping*. Categorical relationships reflect fundamental semantics of the concern categories. *Kind-of* relates (sub)kinds to kinds, *instance-of* relates instances to kinds, *applies-to* relates properties to kinds and instances, *part-of* relates instances to (composite) instances and properties to properties, *member-of* relates physical concerns (instances or collections) to collections, and *relates-to* relates any concern to a topic.

Examples of categorical relationships for concerns in the cache: *logging functionality* is a *kind-of functionality*, *toggle logging* is an *instance-of logging functionality*, *performance applies-to logging functionality*, and *logging functionality* and *logging behavior* *relate-to* the topic *logging*.

Interpretive relationships relate logical concerns with semantics that are not based on categories. One example is *contributes-to*, which indicates that one concern (e.g., *logging behavior*) contributes in some way to another (e.g., *robustness*). Another interpretive relationship is *motivates*, which indicates that one concern (e.g., *robustness*) motivates another (e.g., *logging*).

Physical relationships represent associations among physical concerns (e.g., composition relationships among Java classes in Hyper/J [3]). Mapping relationships represent non-categorical associations between logical and physical concerns (for example, the implementation of a logical function by a Java class).

Predicates represent integrity conditions over various relationships and can be classified accordingly. For example, categorical predicates apply to categorical relationships and interpretive predicates apply to interpretive relationships. One of the former is that no concern can be both a kind and an instance; one of the latter is that if one concern motivates another then the second concern should contribute to the first.

## 4. DISCUSSION

The Cosmos schema is divided into a core part and extensions. The core includes the categories of concerns, relationships, and predicates, as well as the types of concerns and categorical relationships. Particular types of interpretive, physical, and mapping relationships (as well as applicable predicates) are defined as extensions. This is because the core elements are

pertinent in many contexts, whereas the elements addressed by extension tend to be particular to the purpose of concern space modeling or analysis. For example, we have found interpretive relationships useful in analyzing concerns expressed in requirements and design, while physical and mapping relationships have been useful in support of code composition.

A Cosmos model provides a kind of documentation for a software system, but Cosmos is not intended to replace existing forms of documentation. Rather, Cosmos provides an independent representation for the concerns captured in existing work products and relates concerns across work products. Thus, a Cosmos model provides a kind of semantic hyper-index to concerns across a software system and its constituent artifacts.

## 5. ACKNOWLEDGMENTS

Thanks for their helpful comments to Peri Tarr, Harold Ossher, Stefan Tai, Thomas Mikalsen, Judah Diament, Mohamed Kande, Chris Codella, and Nagui Halim.

## 6. REFERENCES

- [1] Clarke, S., Harrison, W., Ossher, H., and Tarr, P. Towards Improved Alignment of Requirements, Design, and Code. In Proceedings, Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA). ACM SIGPLAN Notices, v. 34, n. 10, pp. 325--339, Oct. 1999.
- [2] Harrison, W. and Ossher, H. Subject-oriented Programming (a Critique of Pure Objects). In Proceedings of the Conference on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA), Sep. 1993.
- [3] IBM. Hyper/J. <http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm>
- [4] Iyengar, Arun. *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, Feb. 1999.
- [5] 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. [Note: we expect to add a more up-to-date reference in the future.]
- [6] Merriam-Webster. Collegiate Dictionary on-line, <http://www.m-2.com/>
- [7] 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. Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Minneapolis, Minnesota, Nov. 2000.
- [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.