Aspects, Concerns, Subjects, Views, ... *

Rich Hilliard
Integrated Systems and Internet Solutions, Inc.
150 Baker Avenue Extension
Concord, Massachusetts 01742 USA
rh@isis2000.com
+1 978 318 0000

Introduction

Talk about separation of concerns influences a variety of branches of software engineering. (In fact, it manifests far beyond software engineering in systems engineering, software systems architecture, enterprise and organizational modeling, and other fields, too, but my focus here is software-intensive systems.) As noted in the Call for Participation, concerns underlie a number of other concepts, including those appearing in the title of this paper. It is thus interesting to ask:

> What is a concern?
> What is an aspect?
> What is a view?
> etc. and,
> How do these concepts relate?

In this position paper I ask whether these various notions can be put into a coherent conceptual framework, and I attempt to sketch the beginnings of such a framework. I hope this topic can be a subject[1] for discussion at the workshop, under the category of theoretical foundations.

Aside from academic, or purely conceptual interest, there are a number of reasons why a conceptual framework might be of value.
(1) First, since these notions are relatively recent, people frequently mean different things even in their use of the same terms. Having a common conceptual framework may improve understanding among workers.

*Submission to the OOPSLA'99 Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems

[1]It is hard to write this paper without casually using ordinary terms that already have technical meanings in the context of this workshop (e.g., "subject," "aspect," "role," "viewpoint," ...) – I will disambiguate usage when it is not clear from context.

(2) Building on **(1)**, a common understanding is the basis for sharing and reuse of insights among these different branches of inquiry. For example, work on aspect weaving in object-oriented programming might be applicable to view integration in software architecture – if we have a common frame of reference within which to understand aspects viz a viz views.
(3) A conceptual framework is the basis for formalization, not only for academic interest, but as the basis for building new or improved tools ("tools" includes notations, languages, methods, and automation) for manipulating "concern-oriented" entities. For example, given the widespread use of technologies such as the Unified Modeling Language (UML), what is needed to support concern-orientation in such tools? What constructs are needed? In an earlier paper, I diagnosed the UML [9] for its support for architectural views and viewpoints [3]. Here, I apply these ideas to a wider set of software engineering concerns.

Formalization also makes possible reification – the "open implementation" approach to improving our tools by introducing first-class generalizations of the concepts of interest into them.
(4) Following from **(2)** and **(3)**, there are certain recurring issues (e.g., view declaration, view checking, view integration and traceability) that it may be possible to formulate once and then apply repeatedly – given the right framework in which to pose them.

This work is inspired by the recent IEEE *Recommended Practice for Architectural Description* [5]. Ideas from that work will be noted below.

Concerns

Separation of Concerns begins with Concerns.

What is a concern?

A concern expresses a specific interest in some topic pertaining to a particular system of interest (or other

subject matter).

We have found it useful to articulate concerns in the form of questions [1]:

> How reliable is this system?
>
> What function does the system perform?
>
> How is the system deployed?

Where do concerns come from?

Concerns do not arise in the abstract, but, as the term itself suggests, in relation to (human) interests. In much of the literature, these humans are called *system stakeholders* In system development, stakeholders typically include a client for the system, its users, maintainers, operators, system developers, vendors, and so on [2]. So, we can say, *Stakeholders have Concerns.* Concerns (and stakeholders) arise at all stages of the system life cycle from conception, through requirements, design, implementation, maintenance and evolution. The relation of concerns to stakeholders is many–to–many (see figure 1): a stakeholder may have multiple concerns and a concern may be held by multiple stakeholders. Furthermore, "within" a concern (e.g., reliability), individual stakeholders may have quite different requirements: while User 1 needs 95% reliability, User 2 needs only 50% reliability. Each user is concerned with system functionality, but has very different functional requirements.

Stakeholders have not been dealt with uniformly by existing tools. Special cases of stakeholders (e.g., users) and certain traditional concerns (e.g., functionality) are built into the expressive capability of some modeling techniques. For example, in the UML use case diagram, one models the functional concerns of particular users. But in general, other classes of stakeholders are not well-represented (let alone, reified) in current modeling techniques. In a recent paper, I sketched a modeling ontology for a wider set of stakeholders and architectural concerns [3]. In organizational and process modeling, wider classes of roles [12, 7] have sometimes been modelled.

Views and Viewpoints

A view is a model of a system. Views have a long history in software engineering. (For a "brief history of views" see the Introduction to [4].) What is interesting in this long history is the evolution of *viewpoints* – in earliest work, viewpoints were *fixed items*. For example, structured design methods (circa 1970s–1980s) usually fixed on two viewpoints: the functional viewpoint and the data viewpoint.

In Ross' work on Structured Analysis (RSA, or SADT) viewpoints were considered to be first-class [11]. In RSA, each model had a *model orientation* which declared the purpose, context, and viewpoint of a model. In the present exposition, consider purpose to be the set of concerns the view is intended to address. A model orientation was documented in a textual form. Although informal, this could be considered a primitive form of extension mechanism for viewpoints.

Viewpoints have tended to be informal in most modeling formalisms. However, in the IEEE Recommended Practice for Architectural Description cited above, the architect must *declare* the viewpoint before using it to develop a view. Recent work in Requirements Engineering, also treats viewpoints as first-class entities, with associated attributes and operations [8]. Similarly, the ISO Reference Model for Open Distributed Processing underwrites the construction of individual views with formally defined viewpoint languages [6] – albeit a fixed set.

It seems to me critical to distinguish view from viewpoint (and the analogues of this below) if we are to make progress in conceptualising, formalising and supporting these notions in architecture, requirements, design and implementation languages. Roughly, the view is the instance, and the viewpoint is the reusable (type-like) information that defines the rules for creating and using a view (relative to that viewpoint):

> view : viewpoint :: instance : type

(See [4] for more details on views and viewpoints, in particular a more refined class diagram of their associations, attributes, and methods than the one that appears below.)

When one constructs a view of a system, it is typically intended to address one or more concerns, relative to that system, in the current context. That context may be requirements, design, etc.

Typically, with views, there is an implicit methodological constraint that a view covers the whole system from the selected viewpoint.

Using the Conceptual Framework

Figure 1 summarises the core of the conceptual framework I have in mind. As noted above, it is very much inspired by the IEEE P1471 conceptual model which we developed for understanding the role of architectural descriptions in software-intensive systems.

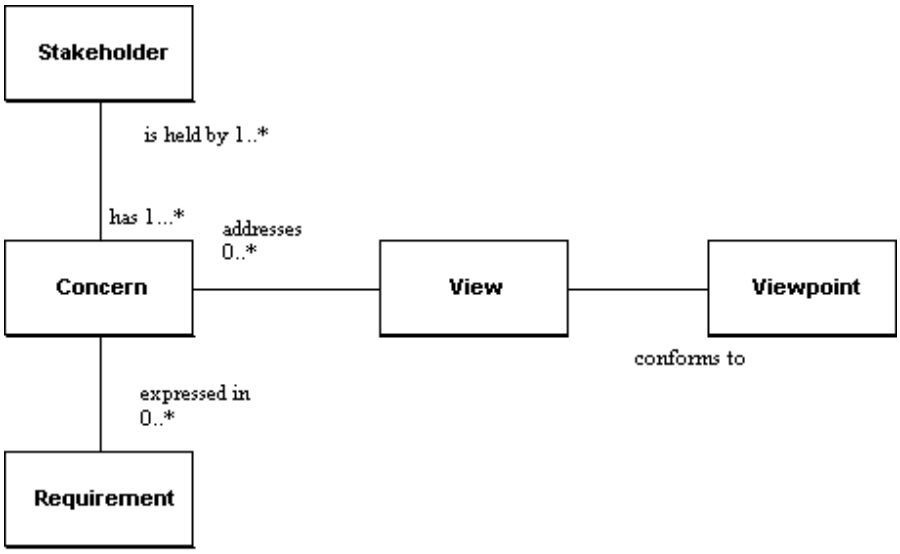

Figure 1: Core of a Conceptual Framework (using the UML syntax)

Let us see how to use the conceptual framework to explain some of the other concepts that are the subject of this workshop.

First, let's look at using the framework to clarify some terminology.

An **aspect**, like a view, addresses a particular concern. Aspects seem to be more "lightweight" than views – without the methodological constraint of covering the whole system. In the current literature on aspects, there has not been attention to separating out what I have called here the concern, the view (model), and the viewpoint, although clearly each exists. If one wants to do checking of individual aspects, it would be useful to do so against an aspect language or aspect type definition.

A **viewpoint**, as that term is used in requirements engineering [8], subsumes both the view (instance) and viewpoint (type) characteristics I have separated here. It describes a set of requirements from the perspective of a single stakeholder.

A **subject** (as in subject-oriented programming) addresses concern with the functionality of object-oriented systems. A subject is a partial view of the system in the context of its domain of application.

Concern spaces, in the spirit of [10], can also be constructed using this conception. A concern space would be formed from the union of all stakeholder concerns. That such spaces are "multi-dimensional, overlapping and interacting" is therefore to be expected.

Second, let's look at defining some recurring notions over the framework.

A view is *well-formed* if it conforms to its viewpoint. It is tempting to allow a view to conform to multiple viewpoints, analogous to multiple inheritance – but I haven't explored this.

Next, let's consider, *What is view integration?*

In this context, two views are *integrated* if a stakeholder's concerns with respect to those views are (all) addressed.

Third, let's return to concerns. At the beginning, we asked where do concerns come from? and we answered, stakeholders. Now we want to look deeper. But, where do stakeholders get concerns from? Some arise from the nature of the problem the system is intended to solve. Others arise from the nature of the technologies selected to solve the problem.

References

[1] David E. Emery, Rich Hilliard, and Timothy B. Rice. Experiences applying a practical architectural method. In Alfred Strohmeier, editor, *Reliable Software Technologies–Ada-Europe '96*, number 1088 in Lecture Notes in Computer Science. Springer, 1996.

[2] Cristina Gacek, Ahmed Abd-Allah, Bradford Clark, and Barry W. Boehm. On the definition of software system architecture. In *Proceedings of the First International Workshop on Architectures for Software Systems*, Seattle, WA, 1995.

[3] Rich Hilliard. Using the UML for architectural description. In *Proceedings of Second Interna-*

tional Conference on the Unified Modeling Language (≪UML≫'99), Lecture Notes in Computer Science. Springer–Verlag, 1999.

[4] Rich Hilliard. Views and viewpoints in software systems architecture. Position paper from the *First Working IFIP Conference on Software Architecture*, San Antonio, 1999.

[5] IEEE Architecture Working Group. *IEEE P1471/D5.0 Information Technology—Draft Recommended Practice for Architectural Description*, August 1999. Available by request from http://www.pithecanthropus.com/~awg/.

[6] International Organization for Standardization. *ISO/IEC 10746 1–4 Open Distributed Processing – Reference Model – Parts 1–4*, July 1995. ITU Recommendation X.901–904.

[7] Philippe Kruchten. *The Rational Unified Process: an introduction*. Addison-Wesley, 1999.

[8] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[9] Object Management Group. *Unified Modeling Language – Notation Guide (version 1.1)*, September 1997. OMG ad/97–08–05.

[10] Harold Ossher and Peri Tarr. Multi-dimensional separation of concerns in hyperspace. Technical Report RC 21452(96717), IBM T. J. Watson Research Center, 1999.

[11] Douglas T. Ross. Structured Analysis (SA): a language for communicating ideas. *IEEE Transactions on Software Engineering*, SE-3(1), January 1977. Also appears in *Programming methodology : a collection of articles by members of IFIP WG2.3* edited by David Gries. New York : Springer-Verlag, 1978.

[12] E. S. Yu and J. Myopoulos. Understanding 'why' in software process modeling, analysis and design. In *Proceedings 16th International Conference on Software Engineering*, Sorrento, Italy, 1993.