Using Aspects in Architectural Description

Rich Hilliard

`r.hilliard@computer.org`

Dedicated to the memory of Douglas T. Ross (21 December 1929 – 31 January 2007), creator of plex and Structured Analysis.

Abstract. This paper sketches an approach to using aspects for architectural description within the conceptual framework of IEEE 1471. I propose a definition of *architectural aspect* within that framework and examine its consequences and motivations. I show that architectural aspects can be accommodated within the current conceptual framework of IEEE 1471 without modification; and outline extensions to the framework which could be candidates for further standardization work, or incorporated into aspect-oriented architectural methods.
Keywords: software systems architecture, architectural description, architectural aspects, architectural viewpoints and views, architectural models

1 Introduction

1.1 Aspect-Oriented Software Development

Work in Aspect-Oriented Programming (AOP) has led to insights in how crosscutting concerns can be captured and implemented – leading to new approaches to modularization of programs which improve program clarity, understandability, modifiability and maintainability. The theme of work in Aspect-Oriented Software Development (AOSD) and the Early Aspects community is to extend these insights throughout all phases of software development to develop "techniques [which] provide systematic means for the identification, modularisation, representation and composition of crosscutting concerns such as security, mobility and real-time constraints" [http://www.early-aspects.net/]. It is hoped that the insights of AOP can provide similar benefits by providing means for managing crosscutting concerns both throughout the software life cycle, and between life cycle phases – if there are uniform concepts of aspects available.

1.2 Aspects in Architecture

Architecting is an important activity of system construction; the term denotes both a specific phase in the life cycle, and a continuing focus on managing certain essential concerns over the life time of a system:

IEEE 1471 defines *architecture* as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [1].

Software systems architects routinely practice separation of concerns and must deal with crosscutting concerns because a primary job of the architect is to insure the quality of a system. While functionality is relatively easy to identify, specify and localize, other qualities (frequently lumped together as simply "non-functional concerns") are pervasive, prevailing architectural concerns. The essence of the architect's job is to negotiate and balance these conflicting concerns of many diverse stakeholders to construct a feasible, functioning system.

The characterization in the previous paragraph is analogous to the situation in programming which motivated the invention of aspects. Perhaps the force of crosscutting concerns is even stronger at the architectural level than in implementation – at least if we take "Boehm's Law" seriously. Qualities dominate the architect's job, whereas functionality is often a consideration which can be deferred to subsequent design and implementation.

Further, this is equally true in systems architecture as in software architecture; so it would be useful to have solutions which are applicable in either domain, since both are frequently parts of major system developments [2].

Therefore, it is worth asking whether there is a role for aspects, or aspect-like constructs, within Architecting. This is a big question and not a new question; it has been raised already in the Early Aspects community. This paper will focus on one topic within the larger question:

How can aspects be used within Architectural Description?

This focus is justified by observing that aspects are largely an approach to modularization, and therefore a syntactic notion. So answers to this question should be offered within the context of existing representational resources. Note that the representational resources for architectural description are quite different from those used in design and implementation. In AOP, aspects crosscut program texts, whereas in modern architectural practices, architectural descriptions are organized using multiple views – can we make sense of aspects crosscutting architectural views? What problems and opportunities does this present?

1.3 Outline for the remainder of paper

In the next section, I outline the conceptual framework for architectural description which forms the basis for IEEE 1471. I go into some detail, because its details make possible the introduction of aspects without modification to the framework. In section 3, I apply the conceptual framework to derive a working definition of architectural aspect and then suggest extensions to the framework which clarify some matters and may be useful for developing aspect-oriented architectural methods. In section 4, I survey related work: the question of the role of aspects in Architecting has been raised by others. I contrast the current approach with other work. The final section contains some conclusions.

2 A Conceptual Framework for Architectural Description

2.1 IEEE 1471

IEEE Std 1471, *Recommended Practice for Architectural Description of Software-Intensive Systems* [1] was the first formal standard to address what is an architectural description (AD). It was developed by the IEEE Architecture Working Group between 1995 and 2000 with representation from industry, other standards bodies and academia, and was subject to intensive reviews by over 150 international reviewers, prior to its publication in 2000.

In 2006, IEEE 1471 became an draft international standard (ISO/IEC DIS 42010) and is now undergoing joint revision by IEEE and ISO (http://www.iso-architecture.org/ieee-1471/). One of the motivations for this paper is to examine what changes are needed in IEEE 1471 to support aspect-oriented architecting.

IEEE 1471 establishes a set of content requirements on an architectural description (any collection of products used to document an architecture) on how ADs should be organized and their information content, while: (*i*) abstracting away from specific media (e.g., text, HTML, XML); (*ii*) being method-neutral (it is being used with a variety of existing and new architectural methods and techniques); and (*iii*) being notation-independent (recognizing that diverse notations are needed for recording various facets of architectures).

IEEE 1471 is built upon a conceptual framework for architectural description (see Figure 1).

The content requirements of an AD per IEEE 1471 are stated using the terms and concepts of the conceptual framework. These rules define what it means for an architectural description (AD) to conform to the Standard. The key concepts needed for discussing architectural aspects are summarized in the remainder of this section.

2.2 Architectures and Architectural Descriptions

I have cited IEEE 1471's definition of "architecture" in the Introduction, and its focus on *architectural descriptions* (ADs): work products used to document the architecture of a software-intensive system.

Although perhaps obvious to some, the conceptual framework distinguishes architectures from architectural descriptions. Architectures are conceptual entities. Architectural descriptions are concrete artifacts.

Architects do their best to express their concepts in a concrete representation which can be captured, managed, reasoned about and shared with others. The force of the standard is on ADs – that which can be concretely captured. Being clear about conceptual entities (architectures) versus artifacts (ADs) will be useful when concerns and aspects are discussed below.

2.3 Stakeholders and Concerns

The audiences for an AD are the various stakeholders of the system.

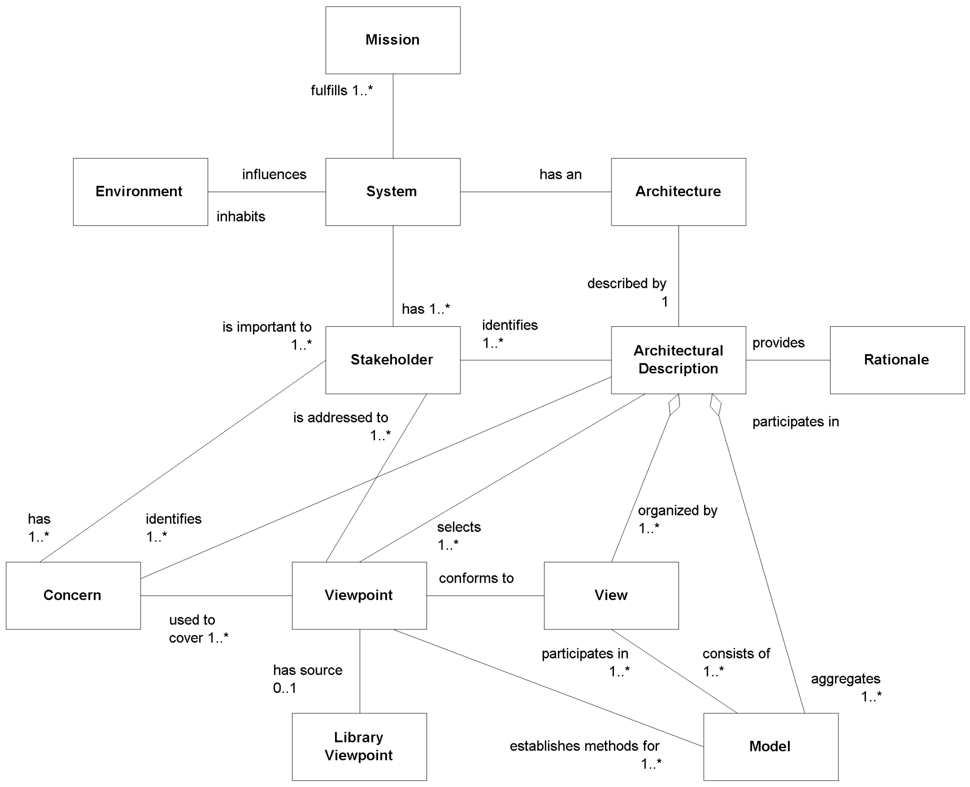


Fig. 1. IEEE 1471 Conceptual Framework

A *stakeholder* is any person, organization or other entity with an interest in the architecture of the system.

The recognition of the role of stakeholders in the AD reflects the multi-dimensional, multi-disciplinary nature of Architecting [4] and facilitates the architectural understanding of their interests with respect to the system.

Within IEEE 1471, those interests are called *architectural concerns* (or *concerns* for short).

Concerns are those stakeholder interests pertaining to the system's development, design, operation, or any other area, which are critical to its architecture. Each concern is an issue which the architecture, and therefore the architectural description, must address. Familiar concerns include: functionality, security, performance, constructibility, reliability – all of which are often considered to be associated with early aspects.

Concerns arise at all stages of a system's construction and operation – it often takes an Architect to recognize which concerns are architecturally significant and cannot be deferred. IEEE 1471 says nothing about the granularity of a concern: it may be as broad as data distribution or as specific as operator security policy.

The sum total of these concerns presents the Architect with a set of problems to solve; the Architect's job is to describe a system manifesting properties that will allow the needs and concerns of the stakeholders for the system to be met.

Therefore, an AD should be explicit in addressing these stakeholders. Under the rules of IEEE 1471, an architectural description must explicitly identify the stakeholders of the system's architecture and enumerate each architectural concern. If an AD does not address all identified stakeholders' concerns it is, by definition, incomplete.

2.4 Architectural Viewpoints and Views

What the Viewer brings to the Viewed in the Viewing, yields the View.
– Douglas T. Ross

It is routine among architectural methods and frameworks to use multiple *architectural views* as a fundamental organizing principle of architectural descriptions to capture information about an architecture (e.g., [5,6,7,8,9,10,11]).

In IEEE 1471, an AD is organized into one or more architectural views. An *architectural view* is defined to be "a representation of a whole system from the perspective of a related set of concerns".

There are many reasons for introducing views, pertaining to separation of concerns and management of complexity (for discussion see [12]). However, early work on architectural views was relatively informal with respect to what constituted a view and how its contents were to be created and analyzed.

Although the use of multiple views was hardly new in IEEE 1471, its contribution is three-fold:

(*i*) to provide conventions for rigorously defining, and therefore using, the contents of a view;

(*ii*) to motivate the selection of views for use through their ability to address specific concerns of specific stakeholders; and

(*iii*) to articulate a "wholeness condition" on views that contributes to the relevance, well-formedness, consistency and completeness of ADs.

In IEEE 1471, the permissible contents of an architectural view is governed by specifying an *architectural viewpoint*.

The idea of a viewpoint first appeared in Ross' Structured Analysis (SADT) [13]. Nuseibeh, Kramer and Finkelstein, working in requirements engineering, treat viewpoints as first-class entities, with associated attributes and operations [14]. The ISO/IEC *Reference Model for Open Distributed Processing* (RM-ODP) defines "*viewpoint* (on a system): a form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system" [15].

All of these precedents contributed to the IEEE 1471 notion:

an *architectural viewpoint* is "a specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis".

Each architectural viewpoint used in an AD must be "declared" before use (either "in line" in the AD or by reference to its source). Table 1 is a template for specifying a viewpoint in accordance with IEEE 1471.

Table 1. Ingredients of a Viewpoint (*based on IEEE 1471*)

A viewpoint is defined by:

- the viewpoint name;
- a set of architectural concerns to be framed by the viewpoint. Views conforming to that viewpoint must address each of those concerns;
- a set of representational resources to be used in constructing conforming views, such as viewpoint languages, modeling techniques and analytical methods; and,
- the source, if any, of the viewpoint (e.g., author, literature citation) if it is defined by reference.

A viewpoint definition may additionally include:

- any formal or informal consistency or completeness checks associated with the underlying method to be applied to models within the view;
- any evaluation or analysis techniques to be applied to models within the view; and
- any heuristics, patterns, or other guidelines which aid in the synthesis of an associated view or its models.

Each architectural view has a governing architectural viewpoint. The viewpoint provides the set of conventions for constructing, interpreting and analyzing a view, including the rules for determining whether it is well-formed.

Each identified stakeholder concern must be framed by at least one of the architectural viewpoints selected for use in an AD; if not, the AD is incomplete.

Making viewpoints first-class provides a means by which the variety of architectural techniques in use today can be uniformly described and therefore compared and perhaps made interoperable.

2.5 Architectural Models

In IEEE 1471, "a view may consist of one or more architectural models". Very little else is said about the use of such models in the 2000 edition of the standard, which has led to some confusion.

There were two motivations for introducing models into architectural views.

First, in order to address a related set of concerns, some views might need to employ more than one type of notation. Per the rules on viewpoint specifications (Figure 1), when a view requires more than one notation, this is captured within the viewpoint definition, specifying the multiple models to be used and the notations for each.

For example, Kruchten's 4+1 Logical viewpoint uses both UML class diagrams and UML component diagrams. These would each be determined in the associated viewpoint definition. Another example could be a Project Management viewpoint specifying that project management views include three types of model: GANTT charts, budgets and org charts to fully address project management concerns.

A second use of architectural models is to capture in one place some architectural detail that contributes to addressing distinct concerns in more than one view – without requiring explicit repetition of that detail in each view.

IEEE 1471 does not provide any further rules or guidance on the use of architectural models within views; this was left open to users of the Standard.

Clearly this latter usage of architectural models is close to the intended purpose of aspects. Architectural models are the essential element of the framework for supporting architectural aspects, as discussed in the next section.

3 Architectural Aspects

The previous section presented a somewhat detailed description of the IEEE 1471 conceptual framework. In this section, I propose a definition of architectural aspect and examine its implications within the IEEE 1471 conceptual framework, demonstrating that aspects can be directly accommodated therein. I then suggest some extensions to the conceptual framework which make its support for architectural aspects clearer, and which are independently motivated for improving that framework. I refer to existing concepts of AOP (including join points, point cuts, advice, base and aspect languages, symmetric and asymmetric paradigms) to motivate the presentation.

Let's begin with a definition. In the remainder of this section, I will examine its consequences and motivations, using the "AOP metaphor" to explore the applicability of aspects to Architectural Description.

Definition. An *architectural aspect* is a shared architectural model addressing exactly one architectural concern.

Model sharing is the architectural analog of the primary motivation for aspects in AOP: to modularize a representation addressing concerns that would otherwise be scattered, and repeated, across multiple views. Sharing is thus crucial in the definition to capturing the crosscutting nature of aspects.[1] If an architectural model is not shared, then by the definition here, it is not an architectural aspect – it is merely a constituent of a view.

One immediate consequence of this definition is that it explains what entities architectural aspects crosscut: they crosscut views and the models which comprise those views.

A second consequence, following from the definitions and rules described in section 2, is that it classifies architectural aspects into:

1. *intra-view* aspects (aspects which apply within a single view); and
2. *cross-view* aspects (aspects which apply across two or more views).

More about these two classes below.

Finally, defining architectural aspects to be essentially a specialization of the existing construct of architectural model is consistent with other practices in AOSD (such as aspect-oriented design) and requires nothing new be added to the IEEE 1471 conceptual framework to support architectural aspects.

Architectural Concerns. Architectural concerns in IEEE 1471 appear to match the idea of concern in AOSD, although not necessarily used in a rigorous manner within that community. (There has been some recent work on concern-based modeling, see 4.)

A concern is a statement of an area of interest on the part of a stakeholder in the architecture of the system. A concern frames a problem to be solved by the architecture. A concern may be large or small, wide or narrow, requirements-, design- or constraint-oriented ... the essence is that it reflects a statement. The

[1] At least within "asymmetric" approaches to aspects, which I will tentatively assume here, although it remains an open question. Under the asymmetric approach, aspects are added to the base – the set of views of the system. For practical reasons, I think Architectural Description will remain view-based because views are frequently organized for an architecture's diverse stakeholders. Adopting the symmetric paradigm would be a radical change for Architectural Description, and would blur the organization of information for stakeholders. This is different from the situation in AOP where, in a sense, there is a single stakeholder: the programmer.

rigor arises as follows. Architectural concerns must be recorded and accounted for in the AD. Each identified architectural concern must be recorded such that an AD is *incomplete* if there are any concerns which have not been framed by at least one viewpoint.

There are concerns of varying granularities: a major concern may warrant a viewpoint of its own (e.g., Functionality, or Distribution). A minor concern might not require its own viewpoint, but could be grouped with others within a viewpoint (e.g., Extensibility of system functionality might be grouped with Functionality).

By limiting an aspect to a single concern in the definition, an aspect becomes the most fine-grained solution element expressible within the framework – without referring to elements of a particular viewpoint language.

It is worth noting that architectural aspects as defined above are quite distinct from aspects that might arise within aspectual (aspect-oriented) viewpoint languages. These are two distinct levels of description. There is no simple relationship between architectural aspects and aspectual languages used as viewpoint languages. Architectural aspects crosscut models and views; whereas aspectual languages within viewpoints would not introduce any crosscutting at an architectural level.

Insights from AOP. Key notions of AOP are join points, point cuts and advice; defined in relation to base and aspect languages. In this section, I briefly discuss the relevance of these notions within Architectural Description, leading up to a proposal at the end of this section.

Unlike the situation in AOP, in Architectural Description there is no single base language; since architectural aspects crosscut views, each viewpoint language is a potential *base language*.[2] Similarly, an architectural aspect could be written in any other viewpoint language chosen to address the identified concern. All languages used in an AD are, by definition, viewpoint languages and must be defined by some viewpoint. Given this, what can we say about base and aspect languages in an architectural setting?

Consider Figure 2, which depicts examples of the two kinds of architectural aspects: intra-view and cross-view aspects. For intra-view aspects, the viewpoint establishes the language to be used for each model. Typically, an intra-view aspect will use the same viewpoint language as the rest of the viewpoint because the concern addressed by the aspect may be subsumed by the concerns framed by the viewpoint – but not necessarily. The aspect may be encapsulating everything there is to say about its identified concern utilizing a language not used elsewhere, but it is still part of a single viewpoint.

For cross-view aspects, the issue of where the aspect language should be defined is not currently addressed by the Standard. There was a notion in IEEE 1471 that each model was: (*i*) defined by exactly one viewpoint: *Viewpoint establishes methods for one or more Models*) and (*ii*) possibly used in other

[2] We could say aspect-oriented architectural description does not enjoy the *Luxury of the Monolinguistic Base* – the flip side of the *Tyranny of Dominant Decomposition*.

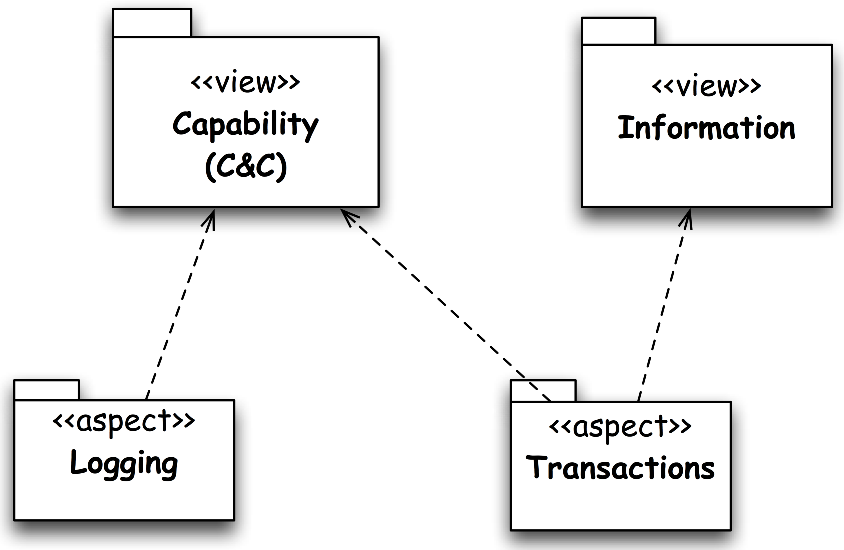


Fig. 2. Examples of intra-view and cross-view architectural aspects

views: *Model participates in one or more Views*); but this was reflected only in the diagram (Figure 1) – there were no rules based on this notion nor any further elaboration. This is an area for improvement in the on-going IEEE 1471 revision.

A *join point* is a location expressed in a base language at which an aspect is to be applied. These will be elements of viewpoint languages.

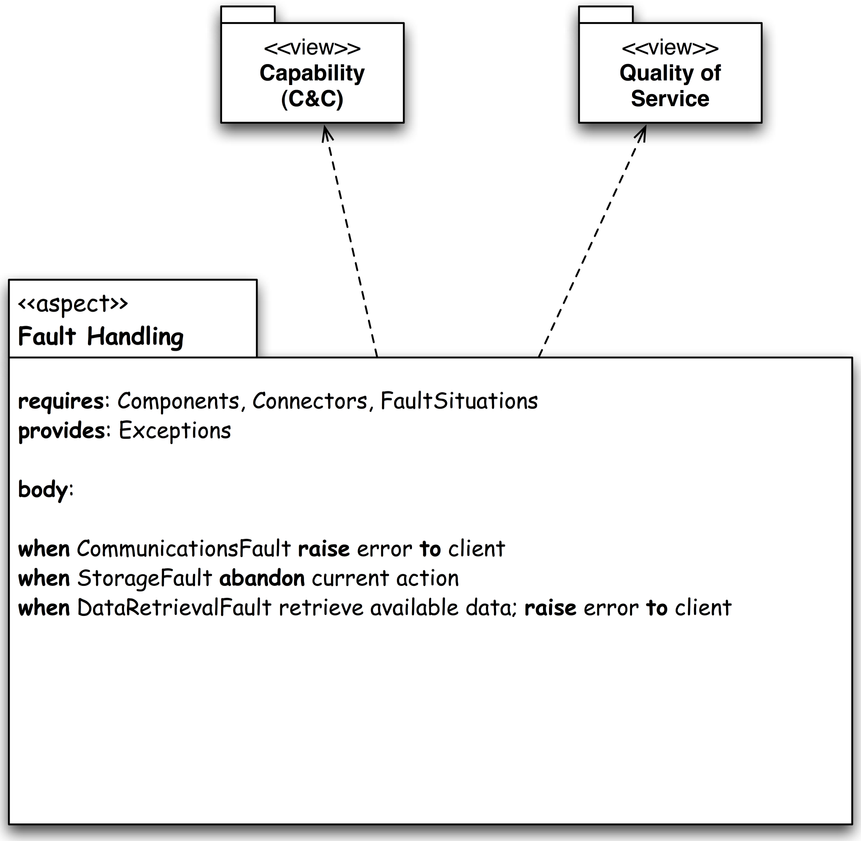
A *point cut* is an expression which picks out a set of join points based on some criteria. Such expressions will be part of the specification of an architectural aspect, and will determine where it applies. This may involve quantification of some kind [16]. *Advice* refers to the rules for combining aspects with base elements to yield the final result.

An architectural aspect can be similarly characterized by its interface (point cut expression) and its body (advice), as proposed below.

Improving Architectural Models. Addressing the considerations of the previous section (analogues of join points, point cuts, advice) for architectural aspects necessarily involves architectural models, since by definition, aspects are a specialization of models. In fact, it can be argued that mechanisms for these are useful independent of aspects. This section will briefly make such a proposal and offer an example.

Currently, these considerations fall under the slogan of *view (or model) integration.* In IEEE 1471, there are no rules pertaining to this; it is an obligation on the end user to specify integration rules as a part of viewpoint definitions. However, with an increased emphasis of models, it might be possible to do better in the revision of the standard, or as a part of aspect-oriented architectural description methods. In particular, there should be a clear solution for models shared across views. Ideally, aspect weaving or composition would be a special case of model integration, but this is an area for future work.

One approach is to revive the provides and requires language of module interconnection to state relations between views and models. In Figure 3, a cross-view aspect, Fault Handling, is depicted. In the UML cartoon, (and in the previous figure), views, models and aspects are denoted by UML packages, with an appropriate stereotype (≪view≫, ≪model≫ and ≪aspect≫, respectively). Arrows show dependencies; such as the application of an aspect to a view. A requires clause captures the element types from a viewpoint language to which an aspect applies (join points). The provides clause captures the element types "delivered" by the aspect. Taken together, the provides and requires clauses establish a contract between models. This is generally useful; not just for specifying aspects for any architectural models and views [17].



<<view>>
Capability (C&C)
<<view>>
Quality of Service
<<aspect>>
Fault Handling
requires: Components, Connectors, FaultSituations
provides: Exceptions
body:
when CommunicationsFault raise error to client
when StorageFault abandon current action
when DataRetrievalFault retrieve available data; raise error to client

Fig. 3. Example: Fault Handling

Another approach could be to introduce a generalized notion of architectural element, as a entity within a view associated with a sort or type defined by its viewpoint, over which to allow quantification.

Under either approach, I suspect universal quantification over types is sufficient for most architectural applications.

View integration in general, the nature of quantification needed, and the investigation of these alternatives, are topics for future work.

Consider[3]:

$model : \_\_\_ :: view : viewpoint$

A viewpoint defines the rules on a view; where are model rules defined? In IEEE 1471 currently, model rules are part of the viewpoint definition (as per Figure 2), but, as noted above, that does not really address the case where a model (whether or not it is an aspect) is shared by two or more views (such as the cross-view aspects introduced above).

Mark Gerhardt has suggested that for the IEEE 1471 revision ___ should be filled by an architectural *model type* (or *metamodel*). A model type would identify the concerns addressed by its model (instances), the model language to be used, and any additional conventions. Viewpoint definitions could then reference these model types. The provides/requires contracts would contribute to defining model types.

4 Related Work

There are three major themes of "related" work – in addition to a few other approaches – which may be classified as follows:

– aspects within a components and connectors-based viewpoint;
– concern-oriented approaches; and
– crosscutting architectural constructs.

Each elucidates some of the issues raised in this paper. I discuss each of these themes in turn.

Aspects in Component-based Viewpoints. A body of work presumes a base representation of components and their connectors, as found in many architecture description languages (ADLs). These include AspectualACME [18], TranSAT [19], Fractal Aspect Component, CAM/DAOP [20], and FuseJ [21]. In these languages, aspects crosscut components and connectors which are elements of the viewpoint language, and the aspects themselves are viewpoint language (ADL) elements. This is in contrast to the present work in which aspects crosscut views. These might better be referred to as component or connector aspects, rather than architectural aspects.

[3] This notation is useful for expressing analogies: $a : b :: c : d$ which is to be read "a is to b as c is to d".

Concern-oriented Approaches. The introduction of architectural concerns into IEEE 1471 arose from attention to *separation of concerns* in software engineering. Recently, concerns have become of increasing interest in their own right within the Aspects community and elsewhere [22,23]. Sutton and Tarr rightly note that concerns are concepts not artifacts; IEEE 1471 focuses on making explicit *identified concerns*; those which have been captured and recorded:

> "If we consider a concern to be any matter of interest in a software system, then concerns subsume aspects as they are usually defined. The commonly accepted definition of aspect (in this context) is a program property that forces crosscutting in the implementation []. This definition recognizes the distinction between concerns (specifically properties) and artifacts (specifically implementations), and it calls attention to an important implementation issue. However, it is relatively narrow (focusing on properties and implementations), it categorizes concerns by their effects (rather than by more intrinsic characteristics), and it links the identification of those concerns to artifacts. In our view, a general notion of concern is needed for AOD (and AOSD in general), one that is not dependent on or linked to notions of artifact." [23]

Approaches to concern modeling could be used within the IEEE 1471 framework, and could lead to additional future requirements in this area for the revision.

Although starting from a different foundation than this paper, Katara and Katz develop several mechanisms which could be of use in the present context [24]. Their work could be used to extend the suggestions here for interpreting join points using the language of **provides** and **requires** for view integration.

Other Crosscutting Architectural Constructs. There are other crosscutting constructs proposed for architectural description that have uses similar to aspects. Two such constructs are architectural perspectives and textures.

Rozanski and Woods (R&W) define perspectives this way [8]: "An *architectural perspective* is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views."

Typically, R&W note, "Applying a perspective almost always leads to the creation of something – usually some sort of model – that provides an insight into the system's ability to meet a required quality property."

A perspective is so close to a viewpoint, which R&W are already using, it is not clear why they needed to introduce it. Like a viewpoint, a perspective defines a set of resources to be used to address a particular set of quality properties (i.e., concerns). Many of their perspectives (Security, Performance and Scalability, Evolution) would just as easily be defined as viewpoints, and R&W's template for perspectives is equivalent to the IEEE 1471 template for viewpoints (although more refined).

Alexander Ran and colleagues have described *architectural textures* as follows:

"Texture is the recurring microstructure of the system. Crosscutting concerns (Kiczales et al., 1997; Ossher and Tarr, 1999) that cannot be localized in a single component must be addressed across multiple components repeatedly. In a well-designed system this must be done in a uniform fashion, which creates recurring microstructure–a definite texture of the system. Examples of crosscutting concerns include exception handling, execution tracing, overload control, flow control, resource reservation policies to mention a few."

That these two constructs have been proposed seems to align with the two types of aspects identified above, as follows: a perspective that crosscuts a defined set of viewpoints could be realized as a cross-view architectural aspect, as defined above. A texture could be captured as an intra-view aspect – the aspect would refer to multiple components at the same time, within a single viewpoint of concern. Treating a texture as an aspect allows the regularity of the "recurring microstructure" to be stated in one place.

Other Approaches. Several papers treat aspects as lightweight views, or do not regard views as essential to architectural description. Bass, Klein and Northrop suggest architectural analogues of join points and point cuts, but equate architectural aspects to architectural views, lacking a rigorous formulation of view [25].

Kim Mens' "Architectural Aspects" [26] is a good antidote to the tyranny of the single-view approach to architecture embodied in early academic work in software architecture, which focused on the components and connectors viewpoint to the exclusion of other viewpoints. However, Mens adopts the symmetric paradigm – equating all architectural elements to aspects. While the choice of paradigm for architectural description is an open question (see footnote 1), it would be a major change to the currently widespread practices of view-based architectural description.

Boucké and Holvoet (B&H) propose extending architectural descriptions with *architectural slices*: "since the architectural elements in a slice are meant to cover a specific driver, there is a direct traceability between drivers and the views describing them (and thus no tangling)" [27]. However, part of the problem B&H set out to solve is self-inflicted – they do not adopt a rigorous notion of view (cf. sections 2, 3 above) and do not use viewpoints to link architectural concerns with the resources used to create views as in IEEE 1471.

Firstly, B&H rightly note the need for stronger connections/relations between elements in distinct views and the relation of decisions and intent, as noted above, need to be better handled in architecture. "Secondly, the standard notion of views does not allow explicit definitions of 'open spots' (like abstract classes or parameters) that should be filled in later." Both of these issues are independent of the use of aspects, and need to be addressed anyway. (See my [28] for an early treatment of these issues.)

All of this could be accomplished within the framework outlined above, making B&H's *slice composition diagram* essentially an architectural viewpoint with matching and other operators to be operations of the associated viewpoint language. Alternatively, these could be defined with reference to an architectural elements metalanguage as suggested above.

5 Conclusions

I have shown how architectural aspects can be accommodated within the current conceptual framework of IEEE 1471 without modification. I have also outlined extensions to that framework which could be candidates for further standardization work, or incorporated into aspect-oriented architectural methods. Some of these are useful refinements of the conceptual framework independent of their relevance to aspect-oriented architectural methods, such as making the handling of architectural models more rigorous and providing an approach to view/model integration. Some of these ideas I hope to explore in the future.

But it is reasonable to ask (as one anonymous reviewer did): *Are aspects needed for architectural description?*

The situation in AD is different from that in programming. The dominant decomposition is multiple viewpoint language-based, and already concern-oriented. Views and models are the modules of an AD, determined by viewpoints, and organized by concerns.

The definition of architectural aspect I've proposed establishes a methodological constraint on when aspects are architecturally relevant within the current conceptual framework of IEEE 1471: when exactly one concern is being addressed; and when that aspect is shared: i.e., when it crosscuts views (or their models). This yields two kinds of aspects which seems to align with existing proposals about textures and perspectives. It also preserves the asymmetric paradigm which seems to inherent to current architectural practice.

Acknowledgments. I would like to thank: Mark Maier (Aerospace) for reading an earlier draft of this paper; the participants in the *First Workshop on Aspects in Architectural Description* (http://aosd.net/workshops/aarch/2007/) for their feedback on some of the ideas here; and the anonymous reviewers of this submission. Their comments made it possible to substantially improve this version.

References

1. IEEE: ANSI/IEEE Std 1471–2000 Recommended Practice for Architectural Description of Software-Intensive Systems. (2000)
2. Maier, M.W., Emery, D., Hilliard, R.: ANSI/IEEE 1471 and systems engineering. Systems Engineering **7**(3) (2004) 257–270
3. : (Ieee 1471 — iso/iec 42010 web site)
4. Maier, M.W., Rechtin, E.: The art of systems architecting. 2nd edn. CRC Press (2000)

5. Kruchten, P.B.: The 4+1 view model of architecture. IEEE Software **28**(11) (1995) 42–50
6. Emery, D.E., Hilliard, R., Rice, T.B.: Experiences applying a practical architectural method. In Strohmeier, A., ed.: Reliable Software Technologies – Ada-Europe '96. Number 1088 in Lecture Notes in Computer Science, Springer (1996)
7. Hofmeister, C., Nord, R.L., Soni, D.: Applied Software Architecture. Addison-Wesley (2000)
8. Rozanski, N., Woods, E.: Software Systems Architecture: Viewpoint Oriented System Development. Addison Wesley (2005)
9. Garland, J., Anthony, R.: Large Scale Software Architecture: A Practical Guide Using UML. John Wiley and Sons (2002)
10. Clements, P.C., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: views and beyond. Addison Wesley (2003)
11. ISO: ISO/IEC 10746-1 Information Technology – Open Distributed Processing – Reference Model: Overview. (1998)
12. Hilliard, R.: Views and viewpoints in software systems architecture. In: First Working IFIP Conference on Software Architecture, San Antonio (1999) Position paper.
13. Ross, D.T.: Structured Analysis (SA): a language for communicating ideas. IEEE Transactions on Software Engineering **SE-3**(1) (1977) 16–34 Also appears in *Programming methodology : a collection of articles by members of IFIP WG2.3* edited by David Gries. New York : Springer-Verlag, 1978.
14. Nuseibeh, B., Kramer, J., Finkelstein, A.: A framework for expressing the relationships between multiple views in requirements specification. IEEE Transactions on Software Engineering **20**(10) (1994) 760–773
15. ISO: ISO/IEC 10746-2 Information Technology – Open Distributed Processing – Reference Model: Foundations. (1996)
16. Filman, R.E., Friedman, D.P.: Aspect-oriented programming is quantification and obliviousness. In Filman, R.E., Elrad, T., Clarke, S., Akşit, M., eds.: Aspect-Oriented Software Development. Addison-Wesley, Boston (2005) 21–35
17. Hilliard, R.: Views as modules. In Balzer, B., Obbink, H., eds.: Proceedings Fourth International Software Architecture Workshop (ISAW-4), 4 and 5 June 2000, Limerick, Ireland. (2000) 7–10
18. Garcia, A., Chavez, C., Batista, T., Sant'anna, C., Kulesza, U., Rashid, A., Lucena, C.: On the modular representation of architectural aspects. In: Proceedings of the 3rd European Workshop on Software Architecture, Nantes, France (2006)
19. Barais, O., Cariou, E., Duchien, L., Pessemier, N., Seinturier, L.: TranSAT: A framework for the specification of software architecture evolution. In: ECOOP First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT04), Oslo (2004)
20. Pinto, M., Fuentes, L., Troya, J.: A component and aspect dynamic platform. The Computer Journal **48**(4) (2005) 401–420
21. Suvée, D., Fraine, B.D., Vanderperren, W.: A symmetric and unified approach towards combining aspect-oriented and component-based software development. In: Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2006). Lecture Notes in Computer Science, Vsters (Stockholm), Sweden (2006)
22. Kandé, M.M.: A Concern-oriented Approach to Software Architecture. PhD thesis, École Polytechnique Fédéral de Lausanne (2003) These n. 2796.

23. Sutton Jr., S.M., Tarr, P.: Aspect-oriented design needs concern modeling. In: Proceedings of AOSD 2002 Workshop on Aspect-Oriented Design. (2002)
24. Katara, M., Katz, S.: A concern architecture view for aspect-oriented software design. Software and Systems Modeling **5** (2006)
25. Bass, L., Klein, M., Northrop, L.: Identifying aspects using architectural reasoning. In: Proceedings Early Aspects Workshop. (2004)
26. Mens, K.: Architectural aspects. In: Proceedings of AOSD 2002 Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. (2002)
27. Boucké, N., Holvoet, T.: Relating architectural views with architectural concerns. In: EA '06: Proceedings of the 2006 international workshop on Early aspects at ICSE, New York, NY, USA, ACM Press (2006) 11–18
28. Hilliard, R., Rice, T.B.: Expressiveness in architecture description languages. In Magee, J.N., Perry, D.E., eds.: Proceedings of the 3rd International Software Architecture Workshop, ACM Press (1998) 65–68 1 and 2 November 1998, Orlando FL.