Using the UML for Architectural Description*

Rich Hilliard

Integrated Systems and Internet Solutions, Inc.
Concord, MA USA
rh@isis2000.com

Abstract. There is much interest in using the Unified Modeling Language (UML) for architectural description – those techniques by which architects sketch, capture, model, document and analyze architectural knowledge and decisions about software-intensive systems. IEEE P1471, the *Recommended Practice for Architectural Description*, represents an emerging consensus for specifying the content of an architectural description for a software-intensive system. Like the UML, IEEE P1471 does not prescribe a particular architectural method or life cycle, but may be used within a variety of such processes. In this paper, I provide an overview of IEEE P1471, describe its conceptual framework, and investigate the issues of applying the UML to meet the requirements of IEEE P1471.
Keywords: IEEE P1471, architectural description, multiple views, viewpoints, Unified Modeling Language

1 Introduction

The Unified Modeling Language (UML) is rapidly maturing into the de facto standard for modeling of software-intensive systems. Standardized by the Object Management Group (OMG) in November 1997, it is being adopted by many organizations, and being supported by numerous tool vendors.

At present, there is much interest in using the UML for *architectural description*: the techniques by which architects sketch, capture, model, document and analyze architectural knowledge and decisions about software-intensive systems. Such techniques enable architects to record what they are doing, modify or manipulate candidate architectures, reuse portions of existing architectures, and communicate architectural information to others. These descriptions may the be used to analyze and reason about the architecture – possibly with automated support. Analyses range from assessing feasibility, *Can the system be built?* [12] to certifying its implementation, *Does the system as implemented conform to the architecture?* [20].

The *Recommended Practice for Architectural Description* (IEEE P1471) [16] represents an emerging consensus for the description of the architectures of software-intensive systems.

* The published version of this paper appears in *Proceedings of ≪UML≫'99*, Lecture Notes in Computer Science, volume 1723, Springer.

In this paper, I investigate the applicability of the UML within the context established by IEEE P1471. I begin with some background on the history and goals for the standard. I then introduce the conceptual framework of P1471. In the main sections of the paper, I review key requirements of IEEE P1471 which pertain to the use of the UML and examine several approaches to using the UML to meet the requirements of P1471. I then address a few issues pertaining to the use of the UML for architectural description which are not specific to P1471. I close with a summary of the key issues and a review of some related work.
Note: This paper is based on version 1.1 of the UML specification which is the currently approved version of the standard. Some changes have been made in version 1.3 of the UML specification, particularly in the area of Model Management, which might affect my analysis. However, at the time of this writing, version 1.3 is not widely available.

2 What is IEEE P1471?

IEEE P1471 is the Draft *Recommended Practice for Architectural Description*.[1] It was developed by the IEEE's Architecture Working Group, chartered and sponsored by the Software Engineering Standards Committee of the IEEE Computer Society. The draft *Recommended Practice* was produced between 1995 and 1998 by a group of approximately thirty participants, and over 140 international reviewers.

2.1 IEEE Goals for P1471

Given the widespread interest in the architecture of software-intensive systems, IEEE recognized the need for providing direction in this area, for both industry and academic application. IEEE set the following goals for the standard:

1. **To take a "wide scope" interpretation of *architecture* applicable to software-intensive systems.** This includes computer-based systems ranging from software applications, information systems, embedded systems, systems-of-systems, product lines and product families – wherever software plays a substantial role in the development, operation, or evolution of a system.
2. **To establish a conceptual framework and vocabulary for talking about architectural issues of systems.** Despite the widespread interest in architecture in both the systems and software engineering communities, there is no common frame of reference for practitioners and researchers in these communities to talk with one another. There are no agreed-upon definitions for terms such as "architecture," "architectural description," and "view."

[1] At the time of this writing, IEEE P1471 has been balloted by the IEEE, and is expected to be approved for use by the time this paper appears. Up-to-date information about IEEE P1471 can be obtained from the IEEE Architecture Working Group (http://www.pithecanthropus.com/~awg).

3. **To identify and promulgate sound architectural practices.** There are already a wide range of software and systems architecture practices. It is a goal of IEEE P1471 to provide a basis on which all of these practices may be defined, contrasted and applied.
4. **To allow for the evolution of those practices as relevant technologies mature.** The IEEE recognized that software systems architectural practices are rapidly evolving, both in industrial use and in the research arena, with respect to architecture description languages, architectural methods, analysis techniques, and architecting processes. It is hoped these practices can be communicated, documented and shared via the framework of P1471. For this reason, the framework should be general enough to encompass current techniques and flexible enough to evolve.

2.2 Using P1471

IEEE P1471 is a *recommended practice* – which is one type of IEEE standard.[2] The important ingredients of IEEE P1471 are:

1. a normative set of definitions for terms including *architectural description*, *architectural view*, *architectural viewpoint*;
2. a conceptual framework which establishes these terms in the context of the many uses of architectural descriptions for system construction, analysis and system evolution; and,
3. a set of requirements on an architectural description of a system.

P1471 applies to *architectural descriptions* (ADs) – any collection of products that purports to describe the architecture of a software-intensive system. An AD is said to *conform* to IEEE P1471 if it meets the requirements of IEEE P1471. Requirements in P1471 are signalled with *shalls*, following usual standards practice. In this way, ADs may be readily checked for conformance to the recommended practice. The requirements of IEEE P1471 are designed to be independent of any individual architectural technique, and therefore should be applicable within a variety of architectural methods and architecture frameworks. It is further hoped that having a common frame of reference will allow greater understanding and sharing between different approaches.

P1471 neither describes nor requires any kind of conformance of systems, projects, organizations, processes, methods, or tools – which are the province of individual methods, frameworks and practicing organizations.

2.3 The P1471 Conceptual Framework

Figure 1, adapted from IEEE P1471, depicts the major conceptual entities referred to by the standard. The conceptual framework is presented as a UML

[2] There are three types of IEEE standard: (i) standards, (ii) recommended practices and (iii) guides.

class diagram. I will further discuss the conceptual framework below, emphasizing elements specific to the UML discussion herein. (For a complete discussion of the framework, please refer to the standard.)

The central abstraction, and primary focus, of the standard is Architectural Description. In P1471, an Architectural Description is a collection of products to document the architecture of a system. P1471 does not specify the format or media for an architectural description. What P1471 *does* specify is certain minimal required content of an AD reflecting current practices and industry consensus.

A key tenet of that consensus is the notion of multiple views. In P1471, an Architectural Description is organized into one or more architectural Views. Most architectural methods and frameworks advocate the use of one or more views of the system as a part of the architectural description. However, the exact views used vary from technique to technique. Rather than require a particular set of views, P1471 leaves this selection to users of the standard.

One of P1471's contributions is to make explicit the notion of an architectural Viewpoint to embody the rules governing a view. It is anticipated that this will allow the definition and reuse of viewpoints, so that varying approaches to architecture may better be able to exchange results, and that in general the growth of the discipline will be facilitated by codifying certain useful patterns of description.

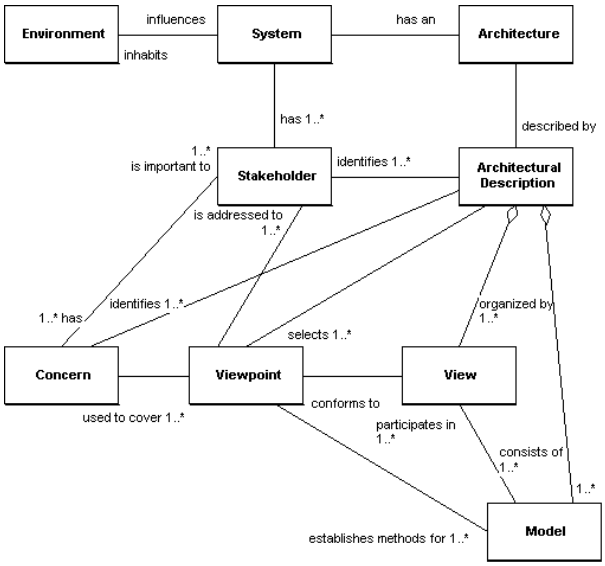

Fig. 1. The P1471 Conceptual Model

IEEE P1471 has been developed to be "notation-independent" – it does not specify any particular notations to be used in an architectural description, leaving this to individual architectural methods or practices. Thus, the question arises, *How does the UML apply in the context of IEEE P1471?* After examining the requirements of IEEE P1471 I will return to this in section 4.

3 P1471 Requirements on Architectural Descriptions

IEEE P1471 establishes requirements on what it means to be a "conforming" architectural description [16]. In this section I highlight those requirements with special implications for the use of the UML. The leading ideas in the IEEE P1471 requirements are **bolded**, and the requirements are paraphrased within the text which follows. For the full set of normative requirements on ADs, readers should refer to the standard.

3.1 Stakeholders and Concerns

As shown in figure 1, P1471 posits two key abstractions in the system environment that influence an architectural description: Stakeholders and Concerns. *Stakeholders* are any individual, class, organization or role that has an interest in the system. Those interests P1471 refers to as Concerns – the architecturally relevant areas of interest on the part of the stakeholders for the system. Concerns capture considerations including the "ilities" (e.g., security, reliability, maintainability) and other system characteristics that influence the architecture.

ADs are interest-relative: A conforming AD identifies the system's stakeholders and their concerns.

In developing the architecture for a system, the Architect will seek to understand who are its stakeholders and what are their concerns. Stakeholders typically include a Client for the system, Users, Maintainers, Operators, System Developers, Vendors, and so on [9]. The stakeholders form an upper bound on the potential audiences for the architectural description.

Concerns are the basis for completeness: In P1471, the content of an architectural description is bounded by the identified concerns of the stakeholders for the system of interest. A conforming AD addresses all stakeholders' concerns identified under the above requirement. An architectural description is incomplete if it does not at least address all such identified concerns. Individual architectural methods may specify additional completeness criteria.

3.2 Architectural Views

P1471 codifies the practice, found throughout software and systems architecture, of using multiple views of a system to document an architecture. Views are recognized as a mechanism to separate concerns, both to reduce perceived complexity and to address the needs of diverse audiences [8].

Multiple views: An AD consists of one or more views. In IEEE P1471, a *view* is a representation of a whole system from the perspective of a related set of concerns.

Views are modular: A view consists of one or more *architectural models*. Each model may use a different representational scheme. In the example below, a capability view is developed using two representational schemes: UML component diagrams and class diagrams.

Inter-view consistency: A conforming AD documents any known inconsistencies among the views it contains. IEEE P1471 does not prescribe any specific consistency-checking techniques between views, which are the realm of individual methods or organizations. It only requires that any inconsistencies, discovered by whatever means, be documented.

3.3 Architectural Viewpoints

The perspective from which a view is constructed is called a *viewpoint*.

Views are well-formed: Each view in a conforming AD is governed by exactly one viewpoint. Viewpoints define the rules for creating and using views. In the IEEE Architecture Working Group, the slogan we have used to relate views and viewpoints is this:[3]

views : viewpoint :: programs : programming language

Concerns drive viewpoint selection: In a conforming AD, each concern is addressed by one or more architectural views. Furthermore, a conforming AD identifies the selected viewpoints and provides the rationale for their selection.

No fixed set of viewpoints: Various architectural methods prescribe a fixed, or starting set of viewpoints, such as Kruchten's 4+1 view model [19], ISO's Reference Model for Open Distributed Processing [17], Siemens [26]. IEEE P1471 does not require any specific viewpoints; leaving this to individual methodological (or religious!) considerations. Instead, P1471 provides mechanisms for insuring that whatever viewpoints are used in a conforming AD, these are documented and understandable. IEEE P1471 does not take a position on where views come from. In the literature, one finds several *stances* toward views [11]. One stance treats views as projective: *a view is a partial projection of a full architectural description*; another stance is constructive: *the architectural description is constructed from one or more separate views*.

Viewpoints are first-class: Each viewpoint used in an AD is declared before use. Like a legend on a map or chart, a viewpoint provides a guide for interpreting and using a view, and appears in a conforming AD together with the view it defines.

[3] The notation $x : y :: z : w$ is read "x is to y as z is to w."

3.4 Viewpoint Example

The remainder of this section presents a brief example of declaring and using a viewpoint conforming to IEEE P1471. The example is drawn from an architectural description for the InternetEngine—a product line architecture for electronic commerce.

Declaring a Viewpoint. IEEE P1471 establishes the minimal information that a conforming AD must contain for each viewpoint used therein. Each viewpoint is specified by:

- the viewpoint name;
- the stakeholders addressed by the viewpoint;
- the stakeholder concerns to be addressed by the viewpoint;
- the viewpoint language, modeling techniques, or analytical methods used; and,
- the source, if any, of the viewpoint (e.g., author, literature citation).

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.

The Capability Viewpoint. This is a brief example of a viewpoint declaration and a resulting view. The Architect of a large, enterprise-wide, distributed information system needs to devise a strategy for the organization of system capabilities and the rules by which those capabilities are constructed and fielded. The "capabilities" are small to mid-sized components intended to encourage reuse across the enterprise and facilitate "plug-and-play" composition.

Viewpoint Name: Capability

Stakeholders: the client, producers, developers and integrators

Concerns: How is functionality packaged? How is it fielded? What interfaces are managed?

Viewpoint language: Components and their dependencies (≪provides≫, ≪requires≫, ≪client-server≫) (using enhanced UML component diagrams); interfaces and their attributes (using UML class diagrams).

Sources: *Also known as:* Static, Application, Conceptual [14]

A Capability View. An application of the capability viewpoint might look like figure 2. The capability view covers all system functionality for operating on data; it is therefore intended as a reference model (template) for new construction and integration. What the cartoon does not show are the assertions associated with each element. Capabilities are constructed using a 5-tier, layered organization with interfaces at each pair of layers. Each layer is a capability. Capabilities can serve other capabilities (horizontal integration). The entire stack is a deployable capability. Rules for interaction among layers, and rules for allowed "content" of a layer are stated in terms of this diagram using OCL. All capabilities must provide certain basic operations, conforming to the Generalized Capability Interface, which is used for the dynamic discovery of capabilities. The Data Access Interface conforms to an XML document type description (DTD). Other interfaces are constrained by well-known, off-the-shelf APIs.

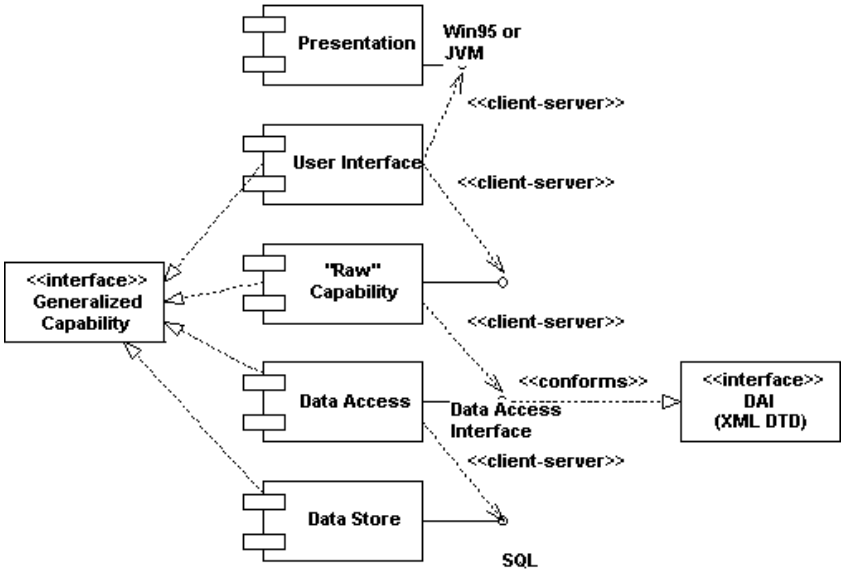

Fig. 2. A Capability View

This capability viewpoint is defined in terms of two existing UML diagram types (class diagrams and component diagrams), stereotype extensions to the component diagram, and certain relations between the diagrams. This ensemble constitutes the *viewpoint language* part of a viewpoint declaration.

Viewpoint Reuse. Unlike a system's stakeholders and its views, viewpoints are not particular to a system. Since they are first-class, it should be possible to keep viewpoints "on-the-shelf" for (re)use. Thus, the architect may be able

to reuse viewpoint descriptions; in IEEE P1471, these are referred to as *library viewpoints*.

So the Architect, instead of defining a capability viewpoint from scratch as above, ought to be able to go to the library and find a viewpoint to meet her needs. One candidate, well-known from Software Architecture, is the structural viewpoint.

Viewpoint Name: Structural

Stakeholders:[4]

Concerns: Define the computational elements of a system and their organization. What elements comprise the system? What are their interfaces? How do they interconnect? What are the mechanisms for interconnection?

Viewpoint Language: Components, connectors, ports and roles, attributes

Analytic Methods: Attachment, type consistency.

Sources: There is general agreement within the field known as Software Architecture, on the usefulness of the ontology of components, connectors, etc. See for example [25]. The ACME architecture description language (ADL) represents a consensus ADL embodying this ontology [10]. See those works for citation of many other ADLs with a similar basis.

4 Using the UML in the Context of IEEE P1471

The UML is nicely suited to P1471 in several ways. Both address a similar scope: software-intensive systems. Like the UML, IEEE P1471 is "intentionally process-independent" [23, §4.2]. It neither defines a life cycle nor an architectural process. Finally, both take as a starting point the need for multiple views in the modeling of software-intensive systems.

Using P1471, the Architect selects architectural viewpoints based on the stakeholders' concerns to be addressed in the architectural description. Ideally, these viewpoints are taken "off-the-shelf" – it is desirable to use predefined viewpoints, when such viewpoints are available. Individual architectural methods and architectural frameworks typically espouse a set of predefined architectural viewpoints (even if they are not referred to as "viewpoints"). For example, the UML *User Guide* [4] advocates a "user-case driven, architecture-centric, iterative, and incremental process" which employs five view(point)s:

> [T]he architecture of a software-intensive system can best be described by five interlocking views. Each view is a projection into the organization and structure of the system, focused on a particular aspect of that system. [4, p31]

There are a range of approaches one might take to applying the UML. In light of the requirements discussed in section 3, we now examine four approaches to using the UML within the context of IEEE P1471.

[4] Not in library, to be filled in at use time.

(1) "Out of the Box" The Architect can adopt the UML as a tool kit of useful notations to be applied to architectural subjects, using it "out of the box." Each of the nine predefined diagram (techniques) is potentially applicable to architectural description. The easiest way to use the UML within the P1471 context is to develop viewpoints which utilize one or more predefined diagram types as viewpoint languages. IEEE P1471 offers a way to understand, and therefore reuse, existing diagram types, as shown in table 1.

Table 1. Predefined UML Diagram Types as Viewpoint Languages

Architectural Viewpoint	**UML Diagram Type(s)**
structural	component diagrams and class diagrams
behavioral	interaction diagrams, activity, state diagrams
user	use case diagrams, interaction diagrams
distribution	deployment diagrams, interaction diagrams

(2) "Lightweight Extension" The Architect can exploit the UML's "lightweight extension mechanisms" (tagged values, stereotypes and constraints) [4] to create new vocabularies for architectural description. Either by creating a UML extension or a variant:

- A *UML extension* is a predefined set of stereotypes, tagged values, and constraints that extend and tailor the UML for a specific domain of application. See [23, §4.1.1]
- A *UML variant* is language built on top of the UML metamodel, specializing that metamodel, without changing any UML semantics. See [23, §4.1.1]

The key point is that such extensions will be developed on a *per-viewpoint* basis – potentially each viewpoint will necessitate its own extension. It would be useful to have a standard way to document viewpoint declarations in the UML, such that they may be notationally depicted, stored and manipulated by tools. Once viewpoints are represented in this form, more interesting uses are possible, such as specialization and combination of viewpoints. It seems such a mechanism needs to be more than what is defined by an extension, since it needs to refer to diagram types; but less than a UML profile [1] – which appears to be much too heavy a mechanism for individual architects, if it has to be applied individually to viewpoints.

(3) "UML as Integration Framework" The Architect, or more likely the architecture team, or firm, might adopt the UML (and its metamodel) as an integrating framework for architectural description. At this level of commitment, one would seek a close correspondence between the P1471 conceptual framework and the UML metamodel.

As noted, the UML and P1471 are *philosophically compatible* on the matter of multiple views; however, while views and viewpoints are "first-class" citizens in the P1471 conceptual framework (figure 1), these concepts appear only informally in the UML specification.[5] For example,

> Every complex system is best approached through a small set of nearly independent views of a model; No single view is sufficient. [23, §4.1.2]

The notion of *viewpoint* appears in the definition of model:

> A *model* is an abstraction of a modeled system, specifying the modeled system from a certain viewpoint and at a certain level of abstraction. A model is complete in the sense that it fully describes the whole modeled system at the chosen level of abstraction and viewpoint. [22, §12.2]

Perhaps closest in spirit to IEEE 1471's notion of a view is the UML notion of a diagram:

> In terms of the views of a model, the UML defines the following graphical diagrams: [use case diagram; class diagram; behavior diagrams: statechart diagram, activity diagram; interaction diagrams: sequence diagram, collaboration diagram; implementation diagrams: component diagram, deployment diagram.]
> These diagrams provide multiple perspectives of the system under analysis or development. ... [23, §4.1.2]

To be precise, we are less interested in individual diagrams than in the rules governing diagram instances. In the UML specification, these are referred to variously as a *diagram type*, a *diagram kind*, or a *diagram technique*. Although there are nine predefined instances of this entity in the UML specification, it has no status in the metamodel either. The "standard diagram types" are defined not in UML itself, but by external rules.

One way to fully support P1471 is to introduce representatives of diagram technique, view and viewpoint into the UML metamodel. This would support both end-user extensiblity by architects and could simplify the specification of the UML. One way to do this is shown in figure 3.

By reifying Diagram Technique we provide a way to document existing diagram types; provide a substrate for differential expression and extensions; and give the end user a means to define new diagram techniques.

Now we may complete the integration of IEEE P1471 and the UML. An architectural description is a model; a model composed of one or more views. Diagram techniques are candidate viewpoint languages which may be referenced in the declaration of viewpoints, and then used to guide the construction of well-formed diagrams which make up the view. A View is one kind of Model,

[5] The entity ViewElement does occur in the UML metamodel (vintage 1.1), but pertains to the graphical presentation (rendering) of ModelElements.

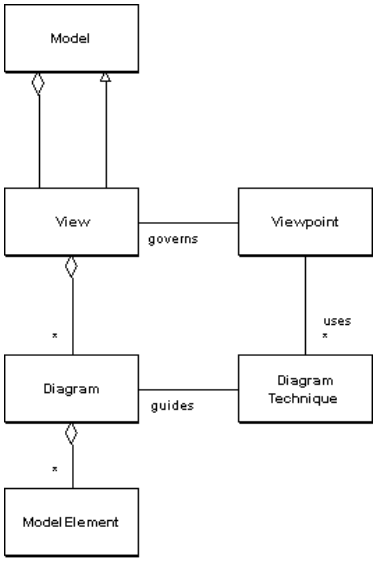

Fig. 3. Enhanced Metamodel Fragment

governed by a well-defined (declared) Viewpoint. A view may be documented with one or more diagrams. Each diagram is guided by the rules of a diagram technique. Diagrams are made up of (owned) ModelElements and references to ModelElements (which may occur in other views).

A useful consequence of the reification of views and viewpoints would be the ability to map out the set of viewpoints employed by an AD and their relationships. A "big picture" notation for this would allow the Architect to sketch and document the expected content of an AD. Figure 4 suggests what a big picture might include: stakeholders (icons), concerns (arrows), viewpoints (triangles) and views (packages). Relationships between packages can be used to express traceability relations and other linkages. In some methods, there is a need to be able to articulate precedence among views – e.g., the requirements view is developed before the design view.

The notions above are useful outside of the realm of architecture. Similar distinctions are useful in requirements and design, as well. A generalized view and viewpoint mechanism would be valuable throughout the UML.

(4) "Outside The UML Ontology"

> The ideal architect should be a man of letters, a skillful draftsman, a mathematician, familiar with historical studies, a diligent student of philosophy, aquainted with music, not ignorant of medicine, learned in the responses of jurisconsults, familiar with astronomy and astronomical calculations.
> — Vitruvius, *De Architectura* (25 BC)

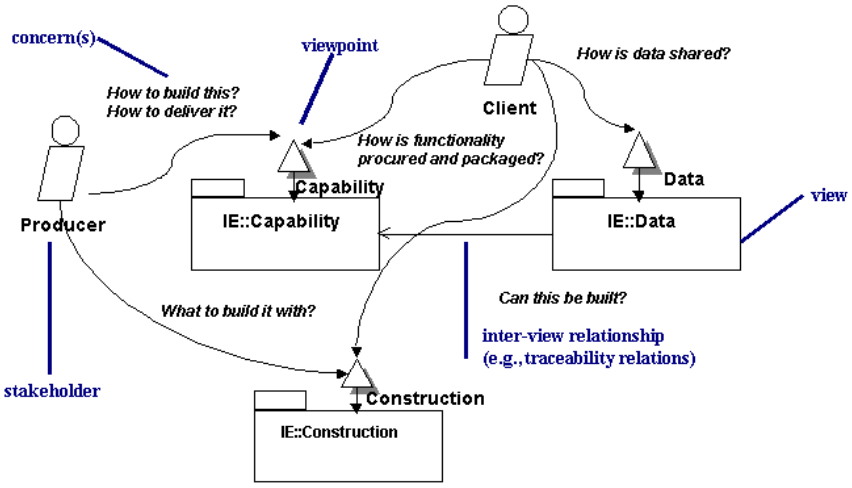

Fig. 4. Fragment of a Big Picture

So far, I have stayed within the subject matters of viewpoints to which the UML is well-suited: the structure, organization and behaviors of a software-intensive system. But architecture – even software-intensive systems architecture – is a multi-disciplinary endeavor [24], and many diverse disciplines may be brought to bear on the Architect's challenge.

There are many existing system disciplines – each with their own existing notations, models and analytic methods. Table 2 suggests a few. For these cases, it is perhaps more efficient to use existing techniques than to recast them in the UML. This does, however, exacerbate the view-integration problem (see section 5).

Table 2. Other Viewpoints

Architectural Viewpoint	**Notations or Techniques**
Management	GANTT charts, budgets, organization charts
Reliability	block models, failure models, ...
Performance	discrete-event models, queueing models ...
Security	(see below)

Information security is a good example. In previous work [6], a security view was developed using a viewpoint language based on the Bell-LaPadula security model [2]. This model provides a viewpoint language along the lines of table 3. Recent approaches adopt other security models e.g., [3]. It would not be efficient to build each such approach into UML.

Table 3. A Security Viewpoint Language (from [6])

(Security) Objects	The resources being protected
(Security) Subjects	Active objects which can perform operations on objects
Policy	The set of rules which specify, for each object and each subject, what operations that subject is allowed to perform on that object
Operations	The ways in which subjects interact with objects
Info domains	Subjects and objects live in domains. Domains may have different security levels, and interconnect with other domains

5 General Issues

There are a number of other issues with the use of the UML for architectural description that are not particular to IEEE P1471. Many of these issues pertain to the generality of the UML ontology of concepts (and its metamodel) – which has been developed for object-oriented analysis and design applications – and the adequacy of that ontology for architectural concerns. For a discussion of expressiveness in architectural description, not specific to the UML, see [13]. In this section, I highlight two issues: view integration and the fixed nature of certain UML "built ins".

View Integration Multiple views necessitate means for view integration: maintaining consistency among views. At present, the UML provides only minimal mechanisms of this kind, such as the trace and refine relationships. The trace relationship is used to represent historical deriviations of an element. The refine relationship is used to represent the same element at different levels of abstraction. The assumption of identity for each of these relations is too restrictive for architectural view integration. Within the P1471 framework, one would like to be able to state such relations as:

1. the *viewpoint level*; e.g., subjects and objects in a security viewpoint are always components in a structural viewpoint;
2. the *view level*: Data access interfaces in the data view are implemented as capabilities (or, components) in the structural view;
3. the *element level*: element a in view $\mathcal{X}$ and element b in view $\mathcal{Y}$ are descriptions of the same thing.

Notations for view integration can be built as relationship stereotypes, to handle the first two cases. For the third case, element-to-element mapping, additional work would be needed, whether in UML, OCL or elsewhere.

Built-In Features Some of the built-in features of the UML, inspired by analogous programming language mechanisms, are not sufficiently general for architectural use, but cannot be overridden in the metamodel, because they are not

first-class. For example, the built-in model of visibility only allows a fixed set of values (public, private, protected). Architecturally, one might like to tailor the visibility model, e.g., to support a particular security model, or for a particular distributed systems paradigm (such as the Distributed Systems Annex of Ada 95).

6 Closing

The UML is well-suited for use with IEEE P1471, providing a ready-made suite of notations to model many of the aspects of the architectures of systems. The predefined diagram techniques are all applicable within commonly used architectural viewpoints (see table 1).

The "lightweight extension mechanisms" – stereotypes, tagged values, and constraints – provide the means to develop specialized vocabularies which the Architect may use as viewpoint languages to address specific areas of concern. For example, the structural viewpoint, well-studied within academic software architecture, is readily captured in this way via component and class diagrams and a set of extensions. Most existing architecture description languages fall within this viewpoint, while other architectural concerns fall outside the current ontology of UML.

By reifying views, viewpoints and diagram techniques, the Architect has the basis to more flexibly specify and manipulate viewpoints than with the proposed profile mechanism. This generality is applicable outside of architecture, as well.

Relation to Other Work. Although first-class views and viewpoints are somewhat new to the architecture community with IEEE P1471, these concepts are found in requirements engineering [7] and aspect-oriented programming [18].

Kruchten was probably the first to use UML for architectural description (even before it was UML!) [19]. Medvidovic and Rosenblum show how to support an architectural style (the C2 style) using the UML's lightweight extension mechanisms [21]. Much of the work on architectural styles in Software Architecture takes place within the Structural Viewpoint. An interesting question is whether the UML can support a "layered" approach to extensions, such that the C2 style "extension" could be defined atop the Structural Viewpoint, as defined above. Hofmeister *et al.* at Siemens define several viewpoints (which they call "views") using UML [15]. As proposed above, each viewpoint uses more than one diagram technique for its expression: their Conceptual Viewpoint comprises UML class, state and sequence diagrams (with the ROOM extensions); their Module Viewpoint comprises class and package diagrams; their Execution Viewpoint comprises class, sequence and state diagrams; and their Code Viewpoint uses component diagrams and a supplementary table [15]. Egyed is using the UML as an integration framework for architectural views [5]. Most work on general ontologies for architecture outside of the system ontology provided by the UML metamodel is qualitative, rather than model-based.

Acknowlegments. This paper is based on a talk given in April 1999 at the *Workshop on Architecture and UML* in Denver, sponsored by Rational. Comments on that presentation by the participants have improved the presentation here. I would also like to thank the members of the IEEE Architecture Working Group for many useful discussions on views and viewpoints. Thanks to the reviewers for useful suggestions and to D. Emery (The MITRE Corporation) for comments on an earlier version of this paper.

References

[1] OMG Analysis and Design Platform Task Force. White paper on the profile mechanism (version 1.0). OMG Document ad/99–04–07, April 1999.

[2] D. Bell and L. J. LaPadula. Secure computer systems: unified exposition and Multics interpretation. Technical Report MTR–2297, The MITRE Corporation, Bedford, MA, 1976.

[3] Christophe Bidan and Valérie Issarny. Dealing with multi-policy security in large open distributed systems. In *Proceedings of the 5th European Symposium on Research in Computer Security*, number 1485 in Lecture Notes in Computer Science, pages 51–66, Belgium, September 1998. Springer-Verlag.

[4] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[5] Alexander Egyed. Integrating architectural views in UML. Technical Report USC/CSE-99-TR-514, Center for Software Engineering, University of Southern California, Los Angeles, CA, 1999.

[6] 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.

[7] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–57, March 1992.

[8] A. Finkelstein and I Sommerville. The viewpoints FAQ. *Software Engineering Journal*, 11(1):2–4, 1996. Also available from ftp://cs.ucl.ac.uk/acwf/papers/viewfaq.ps.gz.

[9] 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.

[10] David Garlan, Robert T. Monroe, and David Wile. Acme: An architecture description interchange language. In *Proceedings of CASCON '97*, pages 169–183, November 1997.

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

[12] Rich Hilliard, Michael J. Kurland, and Steven D. Litvintchouk. MITRE's Architecture Quality Assessment. In *1997 MITRE Software Engineering and Economics Conference*, 1997.

[13] Rich Hilliard and Timothy B. Rice. Expressiveness in architecture description languages. In Jeff N. Magee and Dewayne E. Perry, editors, *Proceedings of the 3rd International Software Architecture Workshop*, pages 65–68. ACM Press, 1997. 1 and 2 November 1998, Orlando FL.

[14] C. Hofmeister, R. L. Nord, and D. Soni. Architectural descriptions of software systems. In D. Garlan, editor, *Proceedings of the First International Workshop on Architectures for Software Systems*, pages 127–137, Seattle, WA, 1995. Published as CMU–CS–TR–95–151.

[15] C. Hofmeister, R. L. Nord, and D. Soni. Describing software architectures with UML. In Patrick Donohoe, editor, *Proceedings of the First Working IFIP Conference on Software Architecture*, pages 145–160. Kluwer Academic Publishers, 1999.

[16] 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/.

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

[18] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. Xerox Palo Alto Research Center, 1997.

[19] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 28(11):42–50, November 1995.

[20] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4), April 1995.

[21] Nenad Medvidovic and David S. Rosenblum. Assessing the suitability of a standard design method for modeling software architectures. In Patrick Donohoe, editor, *Proceedings of the First Working IFIP Conference on Software Architecture*, pages 161–182. Kluwer Academic Publishers, 1999.

[22] Object Management Group. *Unified Modeling Language – Semantics (version 1.1)*, September 1997. OMG ad/97–08–04.

[23] Object Management Group. *Unified Modeling Language – Summary (version 1.1)*, September 1997. OMG ad/97–08–03.

[24] Eberhard Rechtin and Mark Maier. *The art of systems architecting.* CRC Press, 1996.

[25] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an emerging discipline.* Prentice Hall, 1996.

[26] D. Soni, R. L. Nord, and C. Hofmeister. Software architecture in industrial applications. In *Proceedings of the 17th International Conference on Software Engineering*, Seattle, Washington, 1995.