An Ontology of Architectural Design Decisions in Software-Intensive Systems

Philippe Kruchten
University of British Columbia
Vancouver, B.C., Canada
pbk@ece.ubc.ca

Abstract

Architectural design decisions deserve to be first class entities in the process of developing complex software-intensive systems. Preserving the graphs of decisions and all their interdependencies will support the evolution and maintenance of such systems. In this paper we present a possible ontology of architectural design decisions, their attributes and relationships, for complex, software-intensive systems.

Keywords: software architecture, design decisions, design rationale

1. Introduction

The Rational Unified Process® (RUP®) defines software architecture as "the *set of significant decisions* about the organization of a software system: selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization". It also continues saying that "software architecture also involves usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns.[1]" (Kruchten, 1998; IBM, 2003)

For many years we have focused on the result, the consequences of the design decisions we made, trying to capture them in the "design" or the "architecture" of the system under consideration, using often graphics. Representations of software architecture were centered on views, (Clements et al., 2002; Hofmeister, Nord, & Soni, 2000; Kruchten, 1995), as captured by the IEEE 1471 standard (IEEE, 2000), or usage of an architecture description language. But doing so, we lose some of the knowledge that is attached to the decision itself, and to the decision process: rationale, avoided alternatives, etc. In this paper we attempt to make *architectural design decision* a first-class entity in software architecture representation, as suggested by (Bosch, 2004) or (Tyree & Ackerman, 2005). And we, software engineers, are not alone: "In the engineering research community, there is a growing recognition that decisions are the fundamental construct in engineering design" (Chen & al., 2000).

2. Organizing Architectural Design Decisions

We propose in this paper a model of design decisions, where the decision is made a first class entity, in its process context, and not subsumed by a design artifact, such as subsystem, class, process, technology, etc. We propose a classification of design decisions (Section 3), suggest a set of attributes relevant to each class (Section 4), as well as various relationships between decisions (Section 5), and to and from requirements or other development artifacts.

This taxonomy or ontology of design decisions should enable the construction of complex graphs of interrelated design decisions to support reasoning about them.

The description is casually littered with examples, some of them derived from the Canadian Automated Air Traffic Control system (CAATS), documented in (Kruchten & Thompson, 1994; Thompson & Celier, 1995; Sotirovski, 2004).

3. Kinds of Architectural Design Decisions

We've identified 3 major classes of design decisions:

1. Existence decisions,
2. Property decisions, and
3. Executive decisions

[1] Based on an original definition by Mary Shaw, expanded in 1995 by Grady Booch, Kurt Bittner, Philippe Kruchten and Rich Reitman

which we've baptized *ontocrises*, *diacrises* and *pericrises*, respectively[2]. The antonym of an ontocrisis, a ban or non-existence of something, is an *anticrisis*.

3.1. Existence decisions ("ontocrises")

An existence decision states that some element/artifact will positively show up, i.e., will exist in the systems' design or implementation.

There are *structural decisions* and *behavioral decisions.* Structural decisions lead to the creation of subsystems, layers, partitions, components in some view of the architecture. Behavioural decisions are more related to how the elements interact together to provide functionality or to satisfy some non functional requirement (quality attribute), or connectors.

Examples:
- The logical view is organized in 3 layers: Data layer, Business logic layer, User-Interface layer.
- Communication between classes uses RMI (Remote Method Invocation).

Comments:

Existence decisions are not in themselves that important to capture, since they are the most visible in the system's design or implementation, and the rationale can be easily captured in the documentation of the corresponding artifact or element. But we must capture them to be able to relate them to other, more subtle decisions, in particular the alternatives.

3.1.1. Ban or non-existence ("anticrises")

This is the opposite of an existence decision, stating that some element will *not* appear in the design or implementation. They are a subclass of existential decisions in a way.

Comments:

This is important to document precisely because such decisions are lacking any "hooks" in traditional architecture documentation because they are not traceable to any artifact present. Ban decisions are often made as we gradually eliminate possible alternatives.

Examples:
- The system does *not* use MySQL as its relational database system.
- The system does *not* re-use the flight management system from project ASIEW.

[2] From the Greek κρισις, decision and κρινείν, to decide

3.2. Property decisions ("diacrises")

A property decision states an enduring, overarching trait or quality of the system. Property decisions can be design rules or guidelines (when expressed positively) or design constraints (when expressed negatively), as some trait that the system will not exhibit.

Comments:

Properties are harder to trace to specific elements of the design or the implementation because they are often cross-cutting concerns, or they affect too many elements. Although they maybe documented in some methodologies or process in Design guidelines (see RUP, for example), in many cases they are implicit and rapidly forgotten, and further design decisions are made that are not traced to properties.

Examples:
- All domain-related classes are defined in the Layer #2.
- The implementation does not make use of open-source components whose license restricts closed redistribution.

3.3. Executive decisions ("pericrises")

These are the decisions that do not relate directly to the design elements or their qualities, but are driven more by the business environment (financial), and affect the development process (methodological), the people (education and training), the organization, and to a large extend the choices of technologies and tools.

Examples:

Process decisions:
- All changes in subsystem exported interfaces (APIs) must be approved by the CCB (Change Control Board) and the architecture team.

Technology decisions:
- The system is developed using J2EE.
- The system is developed in Java.

Tool decisions:
- The system is developed using the System Architect Workbench.

Comments:

This is likely to be more controversial with the components and connector crowd, but software/system architecture encompasses far more than just views and quality attributes *à la* IEEE std 1471-2000. There are all the political, personal, cultural, financial, technological stuff that sets up huge constraints and all the associated decisions are often never captured or in documents not usually associated with software architecture.

Executive decisions usually frame or constrain existence and property decisions.

4. Attributes of Architectural Design Decisions

See a summary of the attributes in table 1.

4.1. Epitome (or the Decision itself)

This is a short textual statement of the design decision, a few words or a one-liner. This text serves to summarize the decisions, to list them, to label them in diagrams, etc.

4.2. Rationale

This is a textual explanation of the “why” of the decision, its justification. It should be careful not to simply paraphrase or repeat information captured in other attributes, but to have some valued added. If the rationale is expressed in a complete external document, for example, a tradeoff analysis, then the rationale points to this document.

4.3. Scope

Some decision may have limited scope, in time, in the organization or in the design and implementation (see the Overrides relationship below). By default (if not documented) the decision is universal.

Examples:
System scope: The Communication subsystem [is programmed in C++ and not in Java]
Time scope: Until the first customer release [testing is done with Glider].
Organization scope: The Japanese team [uses a different bug tracking system]

4.4. State

Like problem reports or code, design decisions evolve in a manner that may be described by a state machine or a statechart.

Here is an example of such a state machine (cf. also fig. 2):

- **Idea**: Just an idea, captured to not be lost, when doing brainstorming, looking at other systems etc.; it cannot constrain other decisions other than ideas
- **Tentative**: Allows running “what if” scenarios, when playing with ideas.
- **Decided**: Current position of the architect, or architecture team; must be consistent with other, related decisions.
- **Approved**: by a review, or a board (not significantly different than decided, though, in low ceremony organizations).
- **Challenged**: Previously approved or decided decision that is now in jeopardy; it may go back to approved without ceremony, but can also be demoted to tentative or rejected.
- **Rejected**: decision that does not hold in the current system; but we keep them around as part of the system rationale (see Subsumes below)
- **Obsolesced**: Similar to rejected, but the decision was not explicitly rejected (in favour of another one for example), but simply became ‘moot’, irrelevant as a result of some higher level restructuring for example.

This scheme may be too simple for certain environments, or too complicated for others; it has to match a specific decision and approval process. The states can be used to make queries, and as a filter when visualizing a Decision Graph; for example, omit ideas, or display them in green. You would not include the ideas, tentative, and obsolesced decisions in a formal review, for example.

There is an implied “promotion” policy, with the *level* of state being successively:
0: idea and obsolesced
1: rejected
2: tentative and challenged
3: decided
4: approved
And which is used to check consistency of decision graphs (models).

4.5. Author, Time-stamp, History

The person who made the decision, and when the decision was taken. Ideally we collect the history of changes to a design decision. Important are the changes of State, or course, but also change in formulation, in scope, especially when we run incremental architectural reviews.

Example:
“Use the UNAS Middleware”—tentative (Ph. Kruchten, 1993-06-04); decided (Ph. Kruchten, 1993-08-05); approved, (CCB, 1994-01-16); Scope: not for test harnesses; (Jack Bell, 1994-02-01); approved (CCB, 1994-02-27)

4.6. Categories

A design decision may belong to one or more categories. The list of categories is open ended; you could use them as some kind of keywords.

Categories will complement the taxonomy expressed above, if this taxonomy is not sufficient for large projects. (There is a danger in pushing taxonomy too far, too deep too early; it stifles creativity.) Categories are useful for queries, and for creating and exploring sets of design decisions that are associated to a specific concern or quality attribute.

Examples:

- Usability
- Security

But the architects maybe more creative and document:

- Politics: tagging decisions that have been made only on a political basis; it maybe useful to revisit them once the politics change

Example:

"Use GIS Mapinfo" Categories: politics, usability, safety, COTS

4.7. Cost

Some design decisions have an associated cost with them, which is useful to reason about alternatives. For existence decisions, the cost is often that of the development or acquisition of the corresponding design element. For property decisions, the cost is often impossible to estimate. For executive decisions, it varies greatly.

4.8. Risk

Documented traditionally by *Exposure*—a combination of *Impact* and *Likehood* factors—this is the risk associated with taking that decision (see IEEE Std 1540-2001, for example). It is often related to the uncertainty in the problem domain or to the novelty of the solution domain, or to unknowns in the process and organization.

Note that this is different from the risk that a design decision contributed to resolving or mitigating (this mitigation is then part of the rationale).

If the project is using a risk management tool, this should simply link to the appropriate risk in that tool.

Table 1: Attributes of decisions

Name	*Type*
Epitome	Text
Rational	Text or Pointer
Scope	Text
State	Enumeration
History	List of (time stamp+author+change)
Cost	Value
Risk	Exposure level

5. Relationships between Architectural Design Decisions

Decision A *"is Related to"* decision B. See table 2 for a summary.

5.1. Constrains

Decision B is tied to Decision A. If decision A is dropped, then decision B is dropped. Decision B is contingent to decision A, and cannot be promoted higher than decision A.

The pattern is often that a property decision (rule or constraint) constrains an existence decision, or that an executive decision (process, technology) constrains a property decision or an existence decision.

Examples:

"Must use J2EE" constrains "use JBoss"; taking the dotNet route instead of J2EE would make JBoss the wrong choice.

5.2. Forbids (Excludes)

A decision prevents another decision to be made. The target decision is therefore not possible. In other words, decision B can only be promoted to a state higher than 0 if decision A is demoted to a state of 0. (cf. section 4.4)

5.3. Enables

Decision A makes possible Decision B, but does not make B taken. Also B can be decided even if A is not taken. It is a weak form of Constrains.

Examples:

"use Java" enables "use J2EE"

5.4. Subsumes

A is a design decision that is wider, more encompassing than B.

Examples:
"All subsystems are coded in Java" subsumes "Subsystem XYZ is coded in Java"

Note:
Often a tactical decision B has been made, which is later on generalized to A. It is often the case that the design decision could be reorganized to connect relatives of B to A, and to obsolesce B (B can be removed from the graph).

5.5. Conflicts with

A symmetrical relationship indicating that the two decisions A and B are mutually exclusive (though this can be sorted out by additional scoping decisions, cf. 4.3).

Examples:
"Must use dotNet" conflicts with "Must use J2EE"

5.6. Overrides

A local decision A that indicates an exception to B, a special case or a scope where the original B does not apply.

Examples:
"The Comm subsystem will be coded in C++" overrides "The whole system is developed in Java"

5.7. Comprises (is made of, decomposes into)

A high level and somewhat complicated, wide-ranging decision A is made of or decomposes into a series of narrower, more specific design decisions B_1, B_2, … B_n

This is the case in high level existence decisions, where partitioning or decomposing the system can be decomposed in one decision for each element of the decomposition. Or the choice of a middleware system, which implies a choice of various mechanisms for communication, error reporting, authentication etc….

This is stronger than constrains, in the sense that if the state of A is demoted, all the B_i are demoted too. But each B may be individually overridden.

Many of the rationale, alternatives etc. can be factored out and associated with the enclosing decision to avoid duplication, while details on a particular topic are documented where they belong.

Example:
"Design will use UNAS as middleware" decomposes into
"Rule: cannot use Ada tasking" and "Message passing must use UNAS messaging services" and "Error Logging must use UNAS error logging services" and *etc*.

5.8. Is Bound To (strong)

This is a bidirectional relationship where A constrains B and B constrains A, which means that the fate of the two decisions is tied, and they should be in the same state.

5.9. Is an Alternative to

A and B are similar design decisions, addressing the same issue, but proposing different choices. This allows keeping around the discarded choices, or when brainstorming to relate the various possible choices.

Note that not all alternatives are conflicts, and not all conflicts are alternatives. But A conflicts with B can be resolved by making A obsolete and an alternative to B.

5.10. Is Related To (weak)

There is a relation of some sort between the two design decisions, but it is not of any kind listed above and is kept mostly for documentation and illustration reasons.

Examples are high level decisions that only provide the frame for other design decisions, while not being a true constraint (5.1) nor a decomposition (5.7).

5.11. Dependencies

We will say that a decision A depends on B if B constrains A (5.1), if B decomposes in A (5.7), if A overrides B (5.6).

6. Relationship with External Artifacts

6.1. Traces from, and to

Design decisions trace to technical artifacts upstream: requirements and defects, and downstream: design and implementation elements. They also trace to management artifacts, such as risks and plans.

6.2. Does not comply with

It maybe useful to track which portions of the system are not compliant with some design decisions, with a relationship from an artifact in the design or the implementation to a decision.
Example:
Subsystem COMM does not comply (yet?) with decision "no exception are thrown up to the top level unhandled"

Table 2: Relationships

1	Constrains
2	Forbids
3	Enables
4	Subsumes
5	Conflicts with
6	Overrides
7	Comprises (is made of)
8	Is bound to
9	Is an alternative to
10	Is related too
11	Traces to
12	Does not comply with

7. Related work

There has been a lot of work on design rationale in the last two decades; see (Moran & Carroll, 1996) or (Lee, 1997) for an overview. But the focus was on *rationale*, or on supporting the *decision-making process*, not really on the capture, maintenance and further exploitation of large sets of design decisions. There is in particular, the Question, Options and Critera (QOC) approach (MacLean, Young, Belloti, & Moran, 1996), all the work done on Issue-Based Information System (IBIS) by (Conklin & Begeman, 1987) and (Lubars, 1991). Our approach shares most similitude with the Design Rationale Language (DRL) of Jintae Lee (Lee & Lai, 1996).

Ran & Kuusela (1994) looked at design decision trees, which should be a subset of our decision graphs, and which gave us the initial idea behind this paper.

There is a large body of work on decision support in the realm of mechanical, industrial or architectural design, but it is hard to transpose to the design of software. Maybe Nobel Prize winner Herbert Simon (1996) had an explanation: "A characteristic of design that is special to it, besides this gradual emergence of goals, is that the largest task is to generate alternatives. There are lots of theories of decision making, a field that has been heavily cultivated by economists and statisticians. But most theories of decision making start out with a given set of alternatives and then ask how to choose among them. In design, by contrast, most of the time and effort is spent in generating the alternatives, which aren't given at the outset. [. . .] The idea that we start out with all the alternatives and then choose among them is wholly unrealistic."

8. Future work

We plan to validate the model with several non-trivial examples (derived from the design of CAATS for example).

To exploit graphs of design decision, we should establish well-formedness rules of a design decision model, for example:

- No conflict
- All design decisions depend on design decision that are in a higher promotion state
- Bound decisions are in the same state
- All subgraphs depending on a Forbids arc are demoted to level 0.

The next step is the construction of a CASE tool to allow the *manipulation* of design decisions, and the *visualization* of various aspects of such graphs:

- Along a timeline
- Ramifications of a decision
- Cluster of decisions around a category

To support architectural reviews, and the planning of further evolution of a system, as well as for reuse or just education of architects. The tool could also check or enforce the well-formedness rules.

Then there is the validation of the usefulness of the whole idea, by applying it to a few non trivial design endeavours.

Acknowledgements

Thank you to Kosta Beznosov (UBC) and Nicolas Kruchten (UofT) for their comments on an early draft.

9. References

Bosch, J. (2004, May). *Software Architecture: the Next Step*. Paper presented at the First European Workshop on Software Architecture (EWSA 2004), St Andrews, Scotland.

Chen, W., Lewis, K.E., and Schmidt, L. (2000), "Decision-Based Design: An Emerging Design Perspective", *Engineering Valuation & Cost Analysis,* special edition on "Decision-Based Design: Status & Promise", 3(2/3), 57-66.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., et al. (2002). *Documenting Software Architectures: Views and Beyond*. Boston: Addison-Wesley.

Conklin, J., & Begeman, M. L. (1987). gIBIS: a hypertext tool for team design deliberation. Paper presented at the ACM conference on Hypertext, Chapel Hill, North Carolina.

Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied Software Architecture*. Boston: Addison-Wesley.

IBM (2003). *Rational Unified Process*, Version 2003. Cupertino, CA: IBM Rational Software.

IEEE. (2000). *IEEE 1471:2000—Recommended practice for architectural description of software intensive systems.* Los Alamitos, CA: IEEE.

Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Software*, 6(12), 45-50.

Kruchten, P. (1998). *The Rational Unified Process—An Introduction* (1 ed.). Boston, MA: Addison-Wesley.

Kruchten, P., & Thompson, C. J. (1994). *An Object-Oriented, Distributed Architecture for Large Scale Systems*. In Proc. of Tri-Ada'94, Baltimore, November 1994: ACM.

Lee, J. (1997). Design Rationale Systems: Understanding the Issues. *IEEE Expert*, 12(3), 78-85.

Lee, J., & Lai, K.-Y. (1996). What's in Design Rationale. In T. P. Moran & J. M. Carroll (Eds.), *Design Rationale Concepts, Techniques, and Use* (pp. 21-51). Mahwah, NJ: Lawrence Erlbaum Associates.

Lubars, M. D. (1991). Representing Design Dependencies in an Issue-Based Style. *IEEE Software*, 8(4), 81-89.

MacLean, A., Young, R. M., Belloti, V. M. E., & Moran, T. P. (1996). *Questions, Options, and Criteria: Elements of Design Space Analysis*. In T. P. Moran & J. M. Carroll (Eds.), *Design Rationale Concepts, Techniques, and Use* (pp. 53-105). Mahwah, NJ: Lawrence Erlbaum Associates.

Moran, T. P., & Carroll, J. M. (Eds.). (1996). *Design Rationale: Concepts, Techniques, and Use. Mahwah*, New Jersey: Lawrence Erlbaum Associates.

Ran, A., & Kuusela, J. (1996). *Design Decision Trees*. Paper presented at the Eight International Workshop on Software Specification and Design, Paderborn, Germany.

Simon, H. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, Mass.: The MIT Press.

Sotirovski, D. (2004). *An Architecture and Its Rationale*. Richmond, BC: Raytheon (unpublished manuscript)

Thompson, C. J., & Celier, V. (1995, Nov. 6-10). *DVM: An Object-Oriented Framework for Building Large Distributed Ada Systems*. Paper presented at the Tri-Ada'95, Anaheim.

Tyree, J., & Ackerman, A. (2005). Architecture Decisions: Demystifying Architecture. *IEEE Software* (accepted for publication).

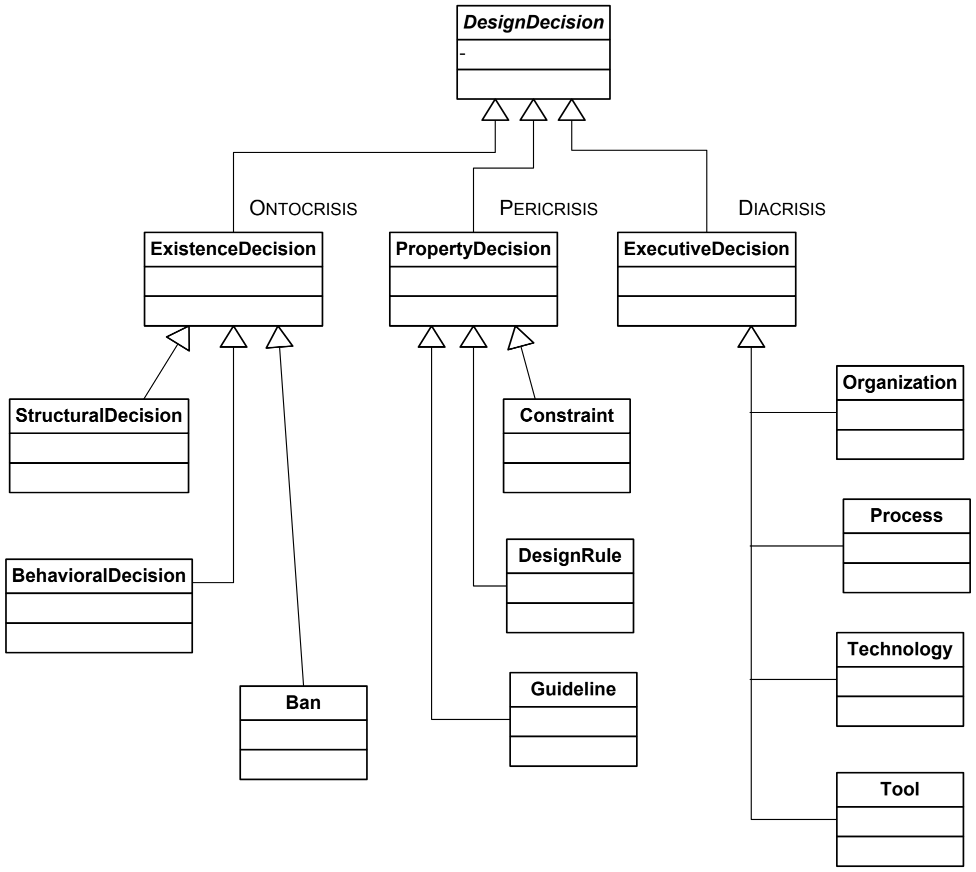

Figure 1: taxonomy of architectural design decisions

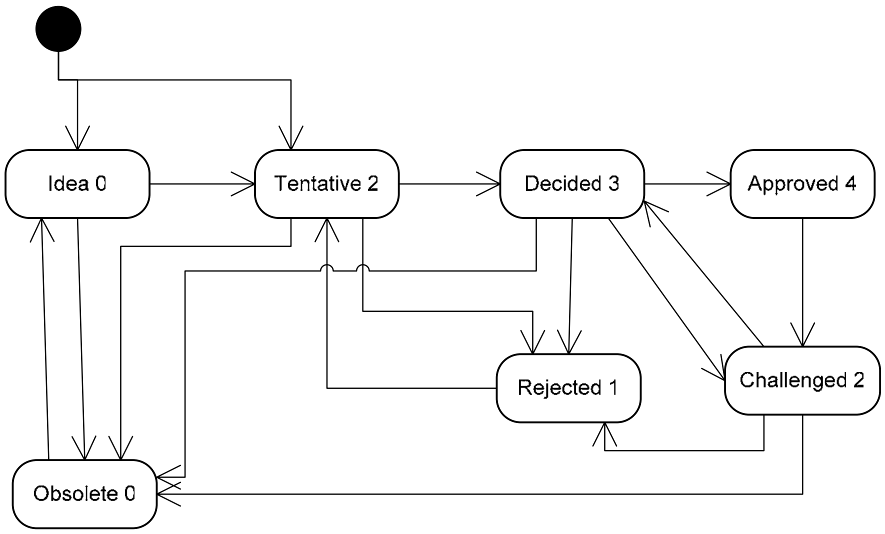

Figure 2: State machine for an architectural design decision