# Modeling Constraints Improves Software Architecture Design Reasoning

Antony Tang
*Swinburne University of Technology,*
*Melbourne, Australia*
*atang@swin.edu.au*

Hans van Vliet
*VU University,*
*Amsterdam, The Netherlands*
*hans@cs.vu.nl*

## Abstract

*Requirements and project-related factors influence architectural design in intricate and multivariate ways. We are only beginning to understand some of the tacit but fundamental mechanisms involved in reasoning about design decisions, and one of them concerns the role of design constraints. This paper examines design constraints and how they shape design solutions. We introduce a design constraint model and an architectural design reasoning process for specifying design constraints and checking for design conflicts. We experiment with using logic for constraint verification with the Alloy tool.*

## 1. Introduction

Functional requirements define the scope of a system and as such narrow down the available design choices. Similarly, non-functional requirements can dictate the available design options. For example, a high performance requirement can eliminate most solutions that cannot meet the required performance level. Other factors, for instance the time and budget for an architect to develop a system, also influence the viability of architectural design. These factors work in aggregation and they jointly constrain or eliminate unviable design options in the architectural design solution space.

Designers consider requirements and other factors when synthesizing a design. They decide on a design which is achievable and will deliver the outcomes that will satisfy the system goals. By using a design constraint as a design reasoning tactic, one could find design options that satisfy the design constraints and eliminate those that do not. Such is the focus of this study. We address three issues relating to design constraints. First, we illustrate the nature of design constraints and the purpose to represent them as a first-class entity. Second, we classify architectural design constraints into four generic categories. Third, we propose a constraint-based design reasoning process for checking design constraints and guiding architects to backtrack their design decisions when design conflicts are detected. We experiment with representing design constraints logically using first order relational logic and in our study we have uncovered inconsistencies in human design reasoning.

## 2. Defining Design Constraints

The current software architecture design practice relies very much on the experience and know how of the designers. Research on design has found that experienced designers just “know” what problems and constraints have to be framed in order to create a viable solution [1]. So the quality of the architectural design relies on the craft of a designer.

In a psychological experiment using Coherence Model of Cognitive Consistency (Co3), the researchers demonstrated that the coherence of decisions can be modeled by a process of constraint satisfaction [2]. It shows that constraint satisfaction plays a key role in the decision making process to connect individual input factors to a common outcome. Similarly in software architecture, experienced architects may intuitively eliminate unviable solutions and steer their decisions towards those viable solutions that meet the design constraints which are often unspecified.

In architectural design, requirements (functional and non-functional) are not the only things that constrain design options in the solution space. Many things constrain the way we design. For instance, business users require software deliveries once every three months in one organization [3]. This constrains on what the development teams can achieve within the time frame. As we make design decisions, technical constraints continue to arise and they influence many other parts of the architectural design.

978-1-4244-4985-9/09/$25.00 ©2009 IEEE


Authorized licensed use limited to: SWINBURNE UNIV OF TECHNOLOGY. Downloaded on January 5, 2010 at 00:09 from IEEE Xplore. Restrictions apply.

Design reasoning involves constraint satisfaction which is more than the satisfaction of requirements and goals, and we need a general **definition** for it: *A design constraint is a limiting factor which specifies the conditions that a viable design solution must satisfy*. We also need a general architectural design constraint **principle**: To design any viable solutions, all design constraints pertaining to the relevant design decisions must be satisfied, allowing constraints changes to be made during the design process.

Design constraints on architectural design can come from different sources and they influence design decisions in different ways. We group constraints into four categories:

- **Requirement Related Constraints** - limiting factors from functional requirements.
- **Quality Requirement Related Constraints** - limiting factors from quality requirements, e.g. performance and availability.
- **Contextual Constraints** – limiting factors that have bearings on the environment for constructing a solution, e.g. project context such as costs and schedule, technology context such as what technology platforms are mandated.
- **Solution-related Constraints** - limiting factors that arise during the design process. They come from technical limitations imposed by the chosen design components.

The first three categories of constraints are mostly dictated by the environment. Constraints from functional requirements are decided and given by the users and the business goals; non-functional requirements constraints come from interpreting the quality requirements; contextual constraints arise from the project and business environments. Solution-related constraints are the side-effects of the chosen solution components. When a design decision is made to employ a certain design component, the design component can constrain how the rest of the system can be designed. The four categories of constraints influence different levels and different parts of an architectural design singly and jointly. Therefore, it is important to identify and specify them in the decision making process and check that they are satisfied to yield a viable design.

**Differentiating design constraints from requirements.** Although requirements constrain design decisions and they are the basis of a software system, requirements and constraints are different. Design constraint is concerned with the limits and the viability of the solution space irrespective of its source. Constraints can arise from a partial solution that impact on the architecture and from different areas and any of the many levels in design and requirements; whereas requirement and goal-oriented methods are concerned mainly with satisfying the goals. During design, the set of design constraints drive the design decisions by: (a) Eliminating unviable design options; (b) Determining conflicting constraints that cannot be satisfied simultaneously at a decision point; (c) A general principle for evaluating design options - allow all relevant constraints to be evaluated in combination.

## 3. Design Constraint Modeling and Design Reasoning

The explicit representation of design constraints can support design reasoning. It allows the checking of design constraint satisfaction for different types of design concern. To allow such checking, design constraints need to be represented either quantitatively or qualitatively:

- **Quantitative design constraint** – specify the constraint, e.g. *x >= 10000 tps*, meaning design option *x* must be able to process at least 10,000 transactions per second. **Design constraint satisfaction** – specify if the behaviors of a design component is satisfied or not, e.g. (*x >= 11000tps and x <= 12000tps*). The behavior of this component is in itself a constraint to other parts of the system.
- **Descriptive design constraint** – specify if the descriptive constraint can be satisfied or not, e.g. a user must login successfully before the user is allowed to carry out any transactions. **Design constraint satisfaction** – All users must be authenticated by the security components (constraint 1) and their privileges checked before any transactions (constraint 2). In a viable design, user's privileges are captured in the *authentication-state* and checked for before any transactions, i.e. the design can satisfy both constraints.

### 3.1. Modeling Design Constraints

In this section, we use an industrial case study to illustrate how design constraints are modeled and how such modeling supports explicit design reasoning.

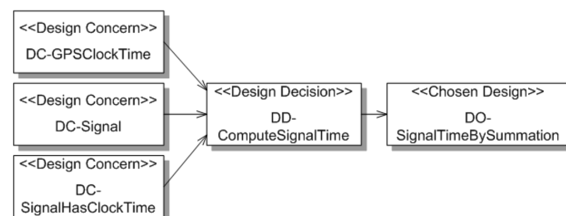


**Figure 1 – A Constraint Modeling Example**



Authorized licensed use limited to: SWINBURNE UNIV OF TECHNOLOGY. Downloaded on January 5, 2010 at 00:09 from IEEE Xplore. Restrictions apply.

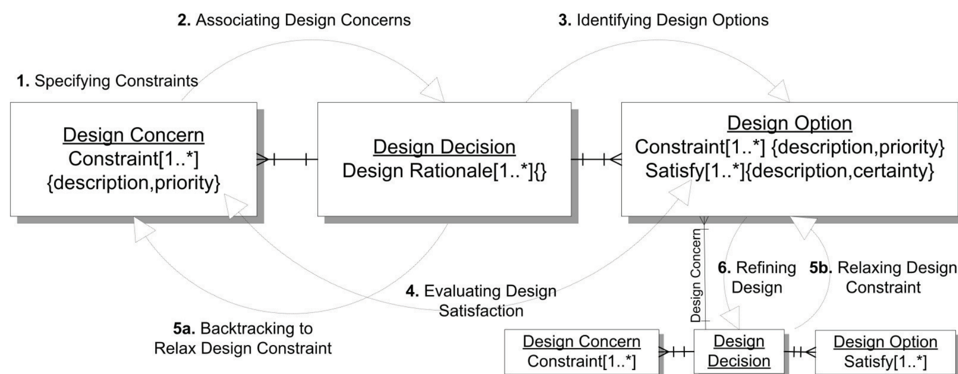


**Figure 2 – Constraint-based Design Reasoning Process**

In this system, monitoring data is collected from a Vehicle Data Logger (VDL) installed in a vehicle. The VDL would stamp each data reading, also called *signals*, with a clock-tick which is relative to a regularly transmitted Coordinated Universal Time (UTC) based on the GPS system. The clock time of the signals are computed by adding a clock-tick to the last GPS clock time (see Fig. 1, the general model is described in [4]).

| | |
|---|---|
| ***Design Concern 1*** - DC-GPSClockTime | |
| Description: | A GPS clock time is available periodically. |
| Constraint: | (A) GPS clock time in UTC format should be available prior to any data readings or signals. |
| ***Design Concern 2*** - DC-Signal | |
| Description: | A signal contains the clock tick relative to the previous GPS clock time. |
| Constraints: | (A) Clock tick is available in each signal. |
| ***Design Concern 3*** - DC-SignalHasClockTime | |
| Description: | All signals should be reported with UTC clock time. |
| Constraint: | (A) Computation of a signal clock time requires the presence of signal clocktick relative to the GPS clocktime. |
| ***Design Decision*** - DD-ComputeSignalTime | |
| Description: | How to compute the clock time of a signal? |
| ***Chosen Design Component 1*** - CD-SignalTimeBySummation | |
| Description: | Compute the clock time of a signal based on the summation of the GPS clock time and the clock ticks in a signal. |
| Satisfaction: | (A) Compute *SignalTime = GPS clock time + clock ticks in signal* |

**Table 1 – Design Constraint Example**

The example in Fig. 1 and Table 1 illustrates the design concerns used in the decision process. Using the three design concerns, a decision was made to create a design object which calculates the UTC time for each signal. During the design process, the design concerns were explicitly represented as requirements and design context, however the design constraints remained implicit. When we explicitly model these constraints, we notice that specific conditions that are required to guide the design are missing from the design concerns. Consider *Design Concern 1* for instance, which states that the GPS clock-time must be available periodically. This outlines *what* is needed, but it does not specify *how* it might affect the design. On the other hand, if we consider how it constrains the design, we note that the clock time must be in a specific format, in this case UTC format, and this reference time must be present before computing the time of any data readings or signals.

### 3.2. Constraints-driven Reasoning Process

Using the industrial case as an illustration, we have observed that (a) design constraints provide specific information for design reasoning; (b) design constraint can be derived from requirements and project context; (c) by representing design constraint we can highlight constraint conflicts when there is no solution that satisfies all related constraints. Through checking constraint satisfaction, designers can identify the design concerns that are conflicting and pinpoint areas where the constraints may be relaxed or compromised. We propose a constraint-driven design reasoning process (Fig 2).

**(1) Specifying design constraints** for each design concern. Each design concern may have 1 or more *design constraints*. **(2) Associating design concerns to a decision** to structure a design decision [5], related design concerns are identified and associated to a decision. **(3) Identifying design options** at each decision point, identify possible design options that satisfy the design concerns and their constraints. During this step, architects specify the *design*



Authorized licensed use limited to: SWINBURNE UNIV OF TECHNOLOGY. Downloaded on January 5, 2010 at 00:09 from IEEE Xplore. Restrictions apply.

*satisfaction* for each design option. **(4) Evaluating design constraint satisfaction** for each design option, evaluate if a design option can satisfy all the *design constraints* by comparing *Design Option Satisfaction* with *Design Constraint.* **(5) Relaxing design constraints** when no design solutions can be found to satisfy a set of constraints at a decision point, one or more design constraints must be relaxed. To do so, architects should traverse backwards to the design concerns to relax their constraints (step 5a in Fig. 2). **(6) Refining design** when the designer is not sure if the chosen design option would work, then further exploration is needed. A designer can repeat steps 1 to 4 to decompose the design to provide more design details until the behavior of the design options are well understood.

### 3.3. Constraint Satisfaction Checking

The checking of constraints based on human interpretation may be error prone. The design solution may not logically fulfill all the constraints. Therefore, it would be desirable to verify the satisfaction of design constraints with a stringent and automated method. To do so, we experiment with first-order relational logic. Firstly, we represent design constraints as *sets* of *elements* and we define the relationships between them, e.g., a *Signal* is a data record containing *clockticks* and *relativeTime*. Secondly, we define the predicates to show *how* the constraints are implemented. Finally, we check if the constraints are violated by the implemented solution. We make use of the Alloy tool to automatically check constraint fulfillment [6]. In Alloy, the signature (*sig*) section defines the design constraints and the predicate (*pred*) section defines how constraints are used by a design object. The following excerpted example represents the constraints and the design outcome described in Table 1:

```
/* Design Concern 1 Constraint A - GPS clocks will be available
and in ascending clock time order */
fact CD_GPSClockTimeInAscendingOrder {
all g: GPSClock - max[GPSClock] | let g' = nextGPSClock[g] |
g'.clocktime >= g.clocktime }
/*  Design concern 2 Constraint A – clocktick is available in each
signal */
sig Signal extends DataRecord {
        clockticks : one Int,
            gpsClock : GPSClock lone -> File,
             relativeTime : Int lone -> Int -> File }
/* Check all signals have a Clock Time */
assert DC_SignalHasClockTime {
            all s: Signal, f: File | let f' = f.next |
              DO_SignalTimeBySummation[f, f', s] =>
         one s.relativeTime.f' }
check DC_SignalHasClockTime for 2 GPSClock,  3 Signal, 2 File
```

An *assertion* acts like a regression test, it tries to discover if there are any scenarios, called *counterexamples*, where the constraints cannot be satisfied by the implemented solution. In this case, Alloy has discovered a counter example. This counter example describes that when the GPS clock time is not available before the arrival of signals, computation cannot proceed. It illustrates that one of the constraints is missing. That is the system does not have a behavior which guarantees that a GPS clock time would arrive before any signals. The Alloy checker has found a logical inconsistency in this design.

## 4. Conclusion

In this paper, we propose to model design constraints as first-class entities. Design constraint can specify *how* requirements and design concerns influence design selection. We suggest that design constraints can come from four different categories of design concerns and they help prune unviable design options at each decision point. Given the representation of a constraint as a first-class entity, it can be used to drive the design reasoning process and verify solution satisfaction. We have proposed a constraint-based design reasoning process that would enable architectural design reasoning and choosing viable design solutions. Designers' reasoning about design constraints may be erroneous, so we experiment with automated checking of architectural design constraint satisfaction. We have used first-order relational logic and the Alloy tool. This method complements the design reasoning process and helps detect logical design errors when constraints are unfulfilled or absent.

## 5. References


[1] N. Cross, "Creative Thinking by Expert Designers," *The Journal of Design Research,* vol. 4 (3), 2004.

[2] K. J. Holyoak and D. Simon, "Bidirectional Reasoning in Decision Making by Constraint Satisfaction," *Journal of Experimental Psychology: General,* vol. 128 (1), pp. 3-31, 1999.

[3] M. R. McBride, "The software architect.," *Communications of the ACM,* vol. 50 (5), pp. 75–81, 2007.

[4] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software,* vol. Mar/Apr 2009 pp. 43-49, 2009.

[5] C. Zannier, M. Chiasson, and F. Maurer, "A model of design decision making based on empirical results of interviews with software designers," *Information and Software Technology,* vol. 49 (6), pp. 637-653, 2007.

[6] D. Jackson, "Automating First-Order Relational Logic," in *Proc. ACM SIGSOFT Conf. Foundations of Software Engineering*, 2000, pp. 130-139.





Authorized licensed use limited to: SWINBURNE UNIV OF TECHNOLOGY. Downloaded on January 5, 2010 at 00:09 from IEEE Xplore. Restrictions apply.