

Motivation

Designing a software system is an explorative, creative process incorporating and balancing the functional and non-functional requirements of a system. Functional requirements have received much attention and have been addressed both structurally and behaviorally. Non-functional requirements have not received as much attention, however, despite the fact that they greatly impact the resulting design of a system.

Non-functional requirements are especially difficult to address due to their unique qualities, particularly (1) their conflicting nature, as can be seen with the trade-off between making a system reliable vs. responsive, (2) their crosscutting nature, as can be seen when incorporating security, which requires modifying a system at multiple points, and (3) their open-ended nature, as can be seen with realizing security, where authentication could be used alone or with encryption. Consequently, correctly designing a system to satisfy non-functional requirements early in the software lifecycle is absolutely critical, since correcting them later can be extremely costly.

Typically, an architect must generate and evaluate many different design alternatives that address non-functional requirements – evaluating them, revisiting them, and refining them, until an adequate design is created.

This is an explorative process, and is not well supported by current techniques. Previous techniques focus instead on documenting the result of this process (e.g., Promela, Rapide ADL, Statecharts, etc.) and analyzing the resulting document (e.g., SPIN, Rapide, Argus-I).

Quality Design with Evaluation of Design Alternatives

Our research goal is to support the explorative design process (Figure 1). As a result, we propose a novel, architecture-based approach that:

1. Explicitly and individually models design alternatives of non-functional requirements. This allows an architect to create and reason about individual design alternatives, rather than modeling each alternative system in its entirety.
2. Composes these design alternatives into a single design using an “architectural weaving” technique. This allows an architect to easily create and maintain different system designs consisting of desired combinations of design alternatives.
3. Analyzes the composed system against specific non-functional requirements. This allows an architect to evaluate different system designs during exploration.

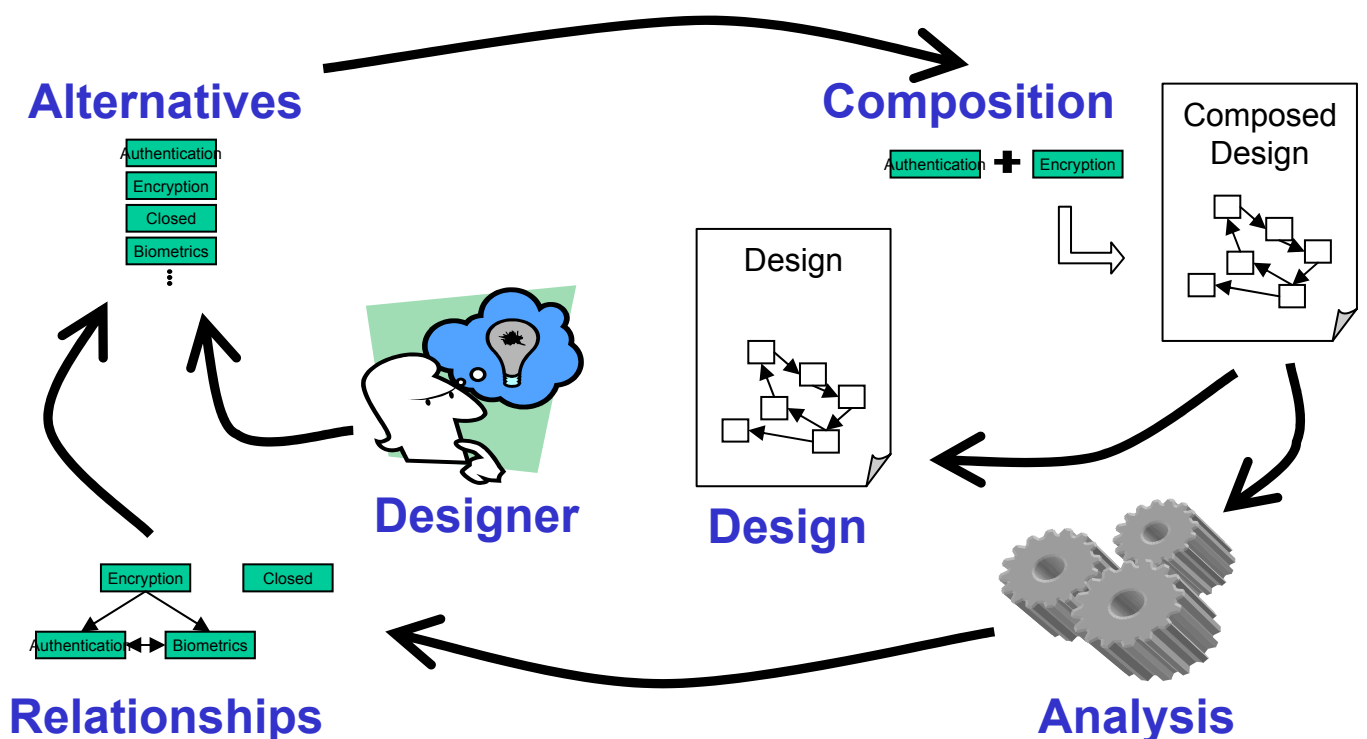
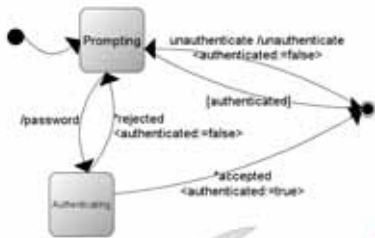


Figure 1: Approach Overview

Authentication Behavior Modifier

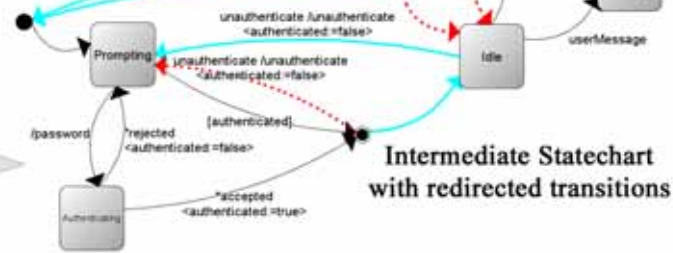


Transitions that initially went from the terminal pseudostate of the behavior modifier are redirected to point from the Idle state of the statechart.

Basic Client Statechart

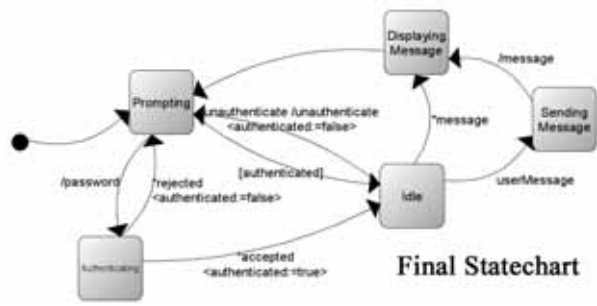
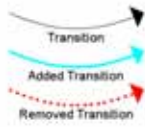


Transitions that initially went to the Idle state of the statechart are redirected to point to the initial pseudostate of the behavior modifier.



Intermediate Statechart with redirected transitions

The initial and terminal pseudostates of the behavior modifier are then collapsed to produce the resulting statechart.



Final Statechart

Figure 2. Weaving the *Authentication* behavior modifier into the *Basic Client* statechart before the *Idle* state

Details of Our Approach

Rather than modeling operationalizable non-functional requirements implicitly within the individual component statechart, we model them explicitly and individually as *behavior modifiers*. Each behavior modifier captures a different behavioral aspect of a non-functional requirement and can be optionally woven into an architecture at key locations specified in *binders*, which is an xml file that describe where behavior modifiers are to be woven into the component statechart of the architecture in terms of trigger events, actions, and states.

The architect chooses which design alternative for non-functional requirements to weave into a system by selecting the design alternatives corresponding binders (Figure 2), then simulates the system to verify that they are being *satisfied*.

By combining different design alternatives into the system and then verifying the system, the architect can easily explore different design alternatives, which was difficult using the traditional approach discussed above since each design alternative had to be specified in its entirety.

How Our Approach Differs

Traditional Approach	Our Approach
Implicit behaviors	Explicit behaviors
Explicit designs	Composed designs
Untangle Alternatives Multiple, Manual Edits	Architecture (Statechart) Weaving
Analyze each Document	Explore & Analyze Compositions Use/Record Relationships

Contact Information

Lihua Xu, Scott Hendrickson

Faculty Advisors: Debra Richardson, Andre van der Hoek

Donald Bren School of Information and Computer Science
University of California
Irvine, California 92697-3425

{lihuax, shendric, djr, andre @ics.uci.edu}