Architectural Prescriptions for Dependable Systems

by
Manuel Brandozzi
and
Dewayne E. Perry

UT ARISE
The University of Texas at Austin
Outline

- Introduction to Architectural Prescriptions
- Overview of Preskriptor and KAOS
- Dependability and Prescriptions
- Current and Future work
What is an Architectural Prescription?

- It’s the architectural design of a system in terms of:
  - The high level components
  - The properties of the components and their interrelationships
  - The constraints (expressed in the vocabulary of the requirements) on the system’s components and on their interrelationships
- An Architectural Prescription provides the very basic framework of the system to satisfy the requirements
Differences between a Prescription and a Description

- A prescription makes the step from requirements to architecture easier to formalize and to perform.
- Problem domain vs. Solution domain
- A prescription enables the use of new, innovative solutions.
- It enables a higher degree of reusability by fewer constraints on the solution space.
KAOS

- The domain is modeled as objects and operations
- First the goals of the overall system (software, people, engines, etc.) are specified
- Then these goals are refined till their sub-goals are assignable to some agents (requirements and assumptions)
Sample Preskriptor Component

- **Component** StockValues [1,1]
- **Type** Data
- **Constraints**
  - Maintain[StoreStockValues],
  - Maintain[AuthorizedAccessesOnly],
  - ...

- **Composed of** DB [1,1], Server [1,1]
- **Uses** MarketConnect to interact with StockMarket
The Preskriptor Process

Step 1
Root Goal(s)
KAOS Objects
Step 2
Root Component(s)
KAOS Goals
Step 3
Potential Sub-component(s)
Architectural Prescription

feedback to requirements
specification phase

from requirements
specification phase
The Preskriptor Process –cont.

From Step 3

- Problem Oriented Prescription
- Solution Oriented Prescription

From Non Problem Domain specifications

- Architectural Goals
- Architectural Styles
- Compatibility Goals

feedback to Step 3
Dependability Requirements

- Non-functional requirements: reliability, performance, reusability, etc.

- Effects:
  - Add new components
  - Transform existing topology
    - i.e. change interactions between components and/or the components’ number of instances
  - Further constrain existing components
Types of Non-Functional Requirements (NFR)

- **Additive NFR**
  - Can be achieved by adding to the system new components and their relationships to the existing components. They may also change the number of instances of the components already in the system.
  - E.g.: fault tolerance

- **Separation NFR**
  - Affect only a subset of the system identifiable by a property
  - E.g.: performance

- **Integral NFR**
  - They either affect the whole system, or it’s not possible to clearly identify the subsystem they affect
  - E.g.: Dynamically reconfigurable
  - Can be achieved by: Dynamic reconfiguration style
ANFR example - before
Some Key Advantages of Our Methodology

- Easiness
- Conformance to requirements
- Separation of concerns
- Customization
- Evolution
- Reusability
Current and Future work

- Validate our methodology with empirical studies
- Find out ways to compositionally transform architectures to achieve the most common ANFRs
- Develop a tool to guide and to automate parts of the requirements to prescription process