Towards Preserving Correctness in Self-Managed Software Systems

Lieven Desmet – Nico Janssens – Sam Michiels
Frank Piessens – Wouter Joosen – Pierre Verbaeten
DistriNet Research Group, Katholieke Universiteit Leuven, Belgium
Lieven.Desmet@cs.kuleuven.ac.be
Overview

- Assumptions
- Motivation
- Research goals
- Approaches
- Open questions
Assumptions

- Adaptive, dynamic software system, with adaptation management within software system

- Adaptivity by typical feedback control loop:
  - sensors
  - gathering data
  - making adaptation decision
  - planning adaptation
  - executing adaptation
DiPS +

Component based, rapid prototyping framework for protocol stacks in highly dynamic network environments
Focus

- **DiPS+ self-management:**
  - external self-management: add-ons to open framework
  - centralized (decision) and distributed (activation)
  - domain specific: flow-based architecture, service based architecture
  - level of adaptation: component/service within a composition, concurrency model
  - how to get it right: MAIN ISSUE

- **application domain:** system software (protocol stacks) and distributed applications
Motivation

Some observations:

- Shift towards component based software design and service oriented software architectures
  - compositions of highly decoupled, reusable, distributed software entities
- Need for evolvability and manageability
  - both at deploy-time and run-time
- Lot of mission critical software systems
  - high availability (24/7)
  - predictable behavior/correctness
Examples

- Implicit dependencies between components
  - e.g. data-centered systems:
    - Central data repository
    - Components can read and write data to the repository
    - Components share data through the shared data repository
Examples

- Composing a data-centered application:
  - Introduces dataflow dependencies between components

Diagram:

- Implicit dependency between components and repository.
Examples

- Replacing a component by a bug-fixed version with additional non-functional support
  - interfaces are stable
  - partially completed transactions
Examples

- Replacing components or services by a faster version
  - interfaces are not stable anymore
  - components are not backward compatible
Research goals

- Consistent software compositions
  - loosely coupled components and services at design-time
  - correctly deployed in different compositions

- Dynamic consistency
  - software consistency before, during and after adaptations for
    - isolated software adaptations
    - distributed software adaptations
Approaches

- Consistent composition
  - ADL's and analysis/verification tools
    - difficult to cover all dependencies/assumptions
  - Support for dataflow dependencies in data centered software systems
    - specific point solution
  - ArchJava bridges gap between ADL and implementation
    - no composition-time information (constraints, dependencies, assumptions) can be used

[Desmet, WADS 2004]
[PhD Aldrich, 2003]
Approaches

➢ Dynamic consistency: Isolated adaptations
  - Achieving a quiescence state in order to have safe software configurations
  - Adding consistency recovery support to capture and reinstate application-state
  - Coexistence of old and new components until converging output (SIMPLEX and HERCULES)

[Magee and Kramer, IEEE Trans. on SE 1990]
[Phd Hofmeister, 1990]
[Cook and Dage, ICSE 99]
[Rivera et al, TR SEI 96]
Approaches

- **Dynamic consistency: distributed adaptations**
  - Consistency preserving distributed adaptations at runtime: Cactus, Ensemble
    - however very specific to a particular application domain and environment
    [Chen et al., ICDCS 02]
    [van Renesse et al., Software – Practice & Experience 1998]
  - NeCoMan
    - encapsulates dynamic consistency in generic and reflective middleware platform
    - only applicable in flow-based architectures
    [Janssens, RM 2004]
  - Lasagne
    - collaborations are composed on a per-request base by selecting appropriate wrappers
    [PhD Truyen, 2004]
Open questions

- Which kind of software systems can benefit of self-managed behavior?
- Is there any need for mission critical, self-managed software systems?
- Can the presented correctness be relaxed in some situations?
- Is fault tolerance complementary to this approach?
Open questions (2)

- What is the unit of adaptation?
- Are adaptations localized or distributed?
- Are adaptations anticipated or unanticipated?

- What is the right balance between formal description/analysis and usability?
- Can tool support completely bridge this gap?