Self-Managed Systems: an Architectural Challenge
Kramer, Jeff and Magee, Jeff
FOSE’07: 2007 Future of Software Engineering

Summarized by Jason Porter
Introduction

• A self-managed system is a system which is capable of:
  ▪ Self-configuration
  ▪ Self-adaptation
  ▪ Self-healing
  ▪ Self-monitoring
  ▪ Self-tuning

• These are often referred to as self-* systems.
Introduction (Cont’d)

• A self-managed software architecture is one in which components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goals of the system.

• The objective is to minimize the degree of explicit management necessary for construction and subsequent evolution whilst preserving the architectural properties implied by its specification.
Why an architectural approach?

- *Generality* – the underlying concepts and principles should be applicable to a wide range of application domains, each associated with appropriate software architectures.

- *Level of abstraction* – software architecture can provide an appropriate level of abstraction to describe dynamic change in a system, such as the use of components, bindings and composition, rather than at the algorithmic level.
Why an architectural approach? (Cont’d)

• *Potential for scalability* – architectures generally support both hierarchical composition and other composition and hiding techniques which are useful for varying the level of description and the ability to build systems of systems, thereby facilitating their use in large-scale complex applications.
Why an architectural approach? (Cont’d)

• *Builds on existing work* – there is a wealth of architecture description languages and notations which include some support for dynamic architectures and for formal architecture-based analysis and reasoning. These provide a good basis for a rigorous approach which could support evaluation and reasoning, constraints and run-time checks.
Why an architectural approach? (Cont’d)

• *Potential for an integrated approach* – many ADLs and approaches support software configuration, deployment and reconfiguration.

• There is existing work on the use of ADLs for software design and implementation from components.
An Architectural “Reference” Model for Self-Management

• Takes inspiration from modern robotic three layer architecture:
  ▪ Control layer (bottom): reactive feedback control
  ▪ Sequencing layer (middle): reactive plan execution
  ▪ Deliberation layer (top): Planning

▪ Accordingly, component control (corresponds to control layer, change management (corresponds to sequencing layer) and goal management (corresponds to deliberation layer).
Component Control

- Consists of the set of interconnected components that accomplish the application function of the system.
- Includes self-tuning capabilities
- Event and status reporting to higher levels
- Operations to support modification – addition, deletion and interconnection of components
Change Management

• Responsible for effecting changes to the underlying component architecture in response to new states reported by the layer below or in response to new objectives required of the system introduced from the layer above.

• Consists of a set of plans which are activated in response to changes of the operating state of the system.

• Can also introduce new components.
Goal Management

• This layer produces change management plans in response to requests from the layer below and in response to the introduction of new goals.

• It takes a declarative specification of system goals, a snapshot of the current state of the system and produces a change plan which moves the system from its current state to a state which satisfies the system goals.
Figure 1 – Three Layer Architecture Model for Self-Management.
Research Challenges (Component Control Layer)

• Ensuring safe application operation during change.
• This involves finding scalable algorithms that minimize disruption during change and ensure safety properties are not violated.

Figure 2 – Example component model

Figure 3 – Example component architecture
Research Challenges (Change Management Layer)

- Dealing with the issue of distribution and decentralization.
- Distribution contributes the problem of obtaining consistent views of system state on which to base change decisions.
- Decentralization of control brings the problem of robust execution in a situation in which partial failure can occur.
Research Challenges (Change Management Layer) Cont’d

- Need for some level of local autonomy while preserving global consistency.
- Problem: view must be complete and requires total order bus to keep views consistent.
Research Challenges
(Change Management Layer) Cont’d

• Proposed solution was not scalable
• We need systems which can accommodate partial inconsistent views and as a consequence relax need for totally ordered broadcast communication.
• The challenge is to find change management algorithms that can tolerate inconsistency and which eventually terminate in a system that satisfies constraints.
Research Challenges
(Goal Management Layer)

• Solutions so far have focused on dealing as far as is possible with planning by designing a set of plans offline that can be shown by construction or by verification processes to satisfy system constraints for a range of possible system states.

• The challenge is to provide an online planner which is invoked by the change management layer when none of the current plans apply to the observed system state.

• This will require an automatic decomposition of goals and the generation of operational plans from these goals.
Conclusion

• An architectural approach offers the required level of abstraction and generality to integrate possible solutions to the challenges posed.
• Defined a three layer architectural model to address these challenges.
• At the component layer, the main challenge is to provide change management which reconfigures the software components, ensures application consistency and avoids undesirable transient behavior.
Conclusion (Cont’d)

• At the change management layer, decentralized configuration management is required which can tolerate inconsistent views of the system state, but still converge to a satisfactory stable state.

• At the goal management layer, some form of constraint-based online-planning technique is required.

• Solutions to these challenges need to be integrated to provide a comprehensive solution with a supporting infrastructure.