Reactive Dynamic Architectures

Jesper Andersson
Department of Computer and
Information Science
Linköpings universitet
S-581 83 Linköping

jesan@ida.liu.se

1. ABSTRACT

Dynamic modifications to architectures include replacing components and re-wiring of connectors. In this position paper we present the manner in which architectures can perform self-governed re-configurations dynamically using rule-based architectural agents. Agents can monitor the architecture and perform simple re-configurations. This paper describes work in progress, and point out interesting areas of research connected to architectural agents.

1.1 Keywords

Software Architecture, Dynamic reconfiguration

2. INTRODUCTION

The software component of complex systems is becoming the most important factor for developing successful projects in different engineering disciplines. Almost every system developed today includes some software that is crucial for overall system behavior. When software is introduced in new product domains, such as consumer electronics, several new problems arise. Additional ones arise in distributed applications where communicating processes distributed over a network and their data constitute the application. These two application domains raise several new interesting questions to be answered by the software architecture [1,2] community. In this paper we focus on these questions wherein changes in the software architecture at run-time is one part of the solution.

When software becomes a more important part of in home-electronic devices, some maintenance tasks become more difficult. The software can still be modified and tested on a single machine at the manufacturer's but when the upgrade is to be distributed and installed the problems become more obvious. The manufacturer either has to recall the devices or the upgrade has to be distributed to the end-user. A better approach is to have the system upgrade itself dynamically. The software downloads and installs the upgrade with minimal effect on the system behavior. This approach requires network connectivity of course. Several devices, such as set-top boxes and cellular phones already have such connectivity and in the future new devices will be added to the group.

In a distributed system the application is no longer a monolith executing on a single machine. The application is certainly more difficult to administer and control. It would be most useful if the application itself had the ability to handle problems, such as load balancing and communication difficulties, itself.

The difficulties involved and several others similar to them, have been solved and there exist applications with this kind of functionality. But we claim that these issues can be handled more efficiently at the architecture level. Re-configuration is about structure (e.g. replacing components and re-wiring connectors) and these structural issues should be handled during architectural design.

Dynamic changes to an architecture (dynamism) [3,4,5] is an active area of research within the software architecture community, but to fully handle the problems involved architectures also must have the ability to react to events and perform architectural changes autonomously. These reactive architectures should have the ability to react on events and perform reconfigurations according to some predefined schema.

3. REACTIVE ARCHITECTURES

In earlier works on description and analysis of architectural structures the focus has been on static architectures. Recently, dynamism has been identified as a focus of research.

Allen et.al. use an extended specification in the Wright language[6] in [4]. Certain work on dynamism in architectures implemented in the message-based C2-style [7] proposes a complementary language for expressing modifications and constraints [8]. A similar approach is used in Darwin [5,9] where a reconfiguration manager controls the required reconfiguration using a scripting language.

If an application is to be able to adapt dynamically without requiring outside intervention, additional functionality supporting this must be provided. A possible solution lies in including special components in the architecture. Another proposed solution involves providing intelligent connectors [Assmann, Uwe. personal communication] in workflow systems. This approach attaches Event/Condition/Action-rules (E/C/A) to the connectors. A connector monitors messages dynamically and adapts to structural changes in the workflow.

These approaches fulfill the important separation of the dynamic reconfiguration behavior from the non-reconfiguration functionality, which is emphasized as being necessary by Allen et.al. in [4]. We take this further and require a separation of the logic used to trigger reconfigurations from the actual functionality performing the modifications. This increases the reusability of certain system components and simplifies understanding. Figure 1 provides a schematic overview of a process for adding the reactive behavior involved. The application independent logic supporting the reactive behavior is configured with application specific code. These parts are then combined with the static application architecture to the final reactive architecture.

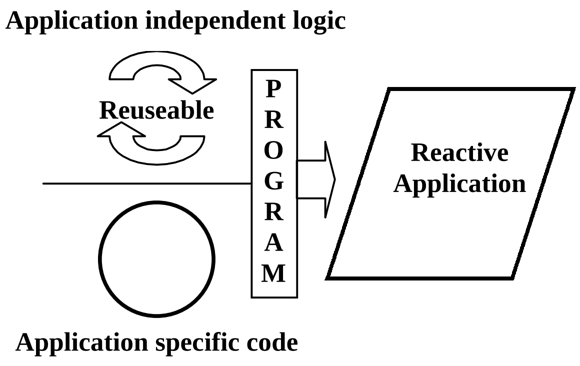

Figure 1 Adding reactive behavior

In our work we use architectural agents to achieve reactive architectures. The agents require functions for dynamic modification and architecture meta-information from the architecture support framework. Agents are rule-based and can manipulate and gather information from the architecture via adapters. An adapter is a domain specific interface that consists of three different classes of methods used by the agent.

1. Effectors
   Effectors are used by an agent when it wants actions to be performed in a specific domain. For instance, one can have an effector that welds a component and a connector in a C2 style specific adapter.
2. Sensors
   Sensors can be used to gather information from the architecture. A sensor can query a component about it's current state. Sensors are used to provide an agent with the sufficient information to make proper decisions.
3. Triggers
   Triggers are ports where the domain can trigger agents. They can be used to notify agents that one or more events have occurred. For instance, one can connect an exception mechanism to an agent via a trigger mechanism.

Adapters for new domains can easily be created. Each adapter implements a common interface. The interface includes functions which the agent uses to find out which effectors, sensors and triggers the adapter supports. An agent can be configured with several adapters, one for each domain. For each agent, a set of rules that describes how the agent should react to certain events has to be developed. In Figure 2 we outline the architecture of a simple agent, configured with two adapters and a rule set.

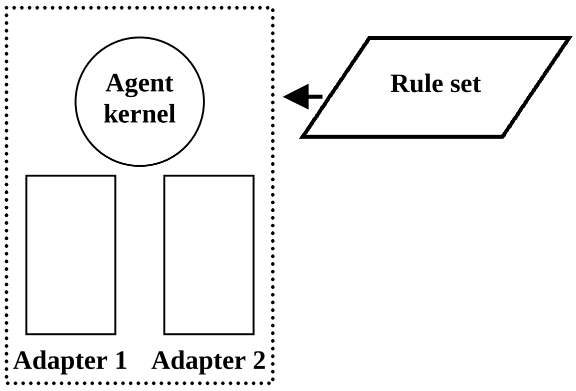

Figure 2 Agent architecture

In our work we have developed a prototype using the IBM-Agent Builder Environment (ABE) [10] and the C2 Java-framework. The IBM-ABE has influenced our work greatly by providing a simple configurable architecture. Especially useful has proven to be the concept of domain specific adapters. Our prototype is a simple component-version-manager where the agent enabled C2 application checks for new versions of components on a remote component server and, if present, downloads and activates these dynamically. To introduce this functionality into the original application we add one single statement wherein we create an agent that supervises the architecture. The remaining functionality lies in the application independent adapters (one adapter for the C2 style architecture and one for the interaction with the component-server) and in the application specific rule-base controlling the agent behavior. In our C2-domain-specific adapter we have included effectors and sensors that reflect the set of commands in the ArchShell [8] command tool.

The component-server adapter includes sensors for querying and effectors to download new versions of components. The prototype and other examples of agent enabled applications is described in [11].

4. CONCLUSIONS and DISCUSSION

Although our work is limited to simple prototypes, we believe that the concept of architectural agents is a major step towards truly reactive architectures. Reactiveness will make applications more autonomous and adaptable to changes in both the application-environment and to internal events which require dynamic reconfiguration.

Having other processes or parts of code monitoring the behavior and perform reconfiguration or communicating with the surrounding environment is not new. But using agents to do this is more flexible and provides a more precise separation of functions for both design and implementation of reactive dynamic architectures.

Several areas are open for more research and numerous questions are involved.

Complex reconfigurations

Can agents be used to perform complex architectural modifications? Is a simple rule-based approach sufficient or must new improved techniques be applied to fully handle these situations? Our perception, based on current experience, is that agents can perform the vast majority of tasks that a system maintainer now usually performs interacting with the system. This thesis must be additionally validated.

Agents in design

Can agents be used during design? Can specifications written in architecture description languages (ADLs), such as ACME[12], be annotated with agents and simulated, triggering agent events and display the resulting modifications in animated form? If possible, this approach will improve the confidence that the developers have in the agent rule-base and reveal problems connected to the reactive behavior early in the design phase.

Implementing functional qualities

Can reactive behavior be used to partially handle some of the functional quality requirements? Lately, several proposals on how this can be achieved have been published. Filman [13], argues that many “ilities” (e.g. Security and Reliability) can be achieved by systematically controlling inter-component communication. Other work proposes specialized components [14] that partially implement some quality aspects of a system. In comparison to our work, we believe that agents can be one way of introducing this kind of behavior in the application architecture. Agents can use sensors to monitor the state of the architecture, a specific component, or a connector. The information derived can be used to make appropriate decisions. Agents can be used to restart components, modify faulty data, re-wire connectors to secondary server’s etc. Another idea is to take advantage of the trigger points where the application (architecture) triggers the agent. The agent can be triggered by specific exceptions generated in the application. In our prototype, we have not explored this possibility but it is possible and we believe it will be useful in certain situations, e.g. one wherein a time-out exception triggers a reconfiguration event in the agent.

In our current work, we focus on how agents can be used during design. The concept involves using one adapter during design that is specific to the ADL used, e.g. an ACME adapter. Rules are developed and tested during design. Later, in the development process during the implementation phase, the agents are introduced to another adapter specificly developed for the run-time support, e.g. a C2 domain adapter. Using this approach, we can develop and test the rule-set early in the development phase.

5. ACKNOWLEDGEMENTS

This material is based upon work sponsored by the Swedish National Board for Industrial and Technical Development (NUTEK).

6. REFERENCES

[1] Shaw, M. and Garlan, D. *Software Architecture, Perspectives on an emerging Discipline.* Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1996.

[2] Perry, D.E. and Wolf, A.L Foundations for the study of Software Architecture. *Software Engineering Notes*, 17(4):40, Oct. 1992.

[3] Medviovic, N. ADLs and Dynamic Architecture Changes. In *Joint Proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96).* pages 24-27, ACM-Press, 1996.

[4] Allen, R.J., Douence, R. and Garlan D. Specifying dynamism in software architectures. In *Proceedings of Foundations of Component-Based Systems Workshop*, Sep. 1997,http://www.cs.iastate.edu/~leavens/FoCBS/

[5] Kramer, J. and Magee, J. Dynamic structure in software architectures. In *Proceedings of the fourth ACM SIGSOFT symposium on Foundations of software engineering (FSE'96)*, pages 3-14. ACM-Press, October 1996.

[6] Allen, R.J. *A Formal approach to Software Architecture,* PhD thesis, 1997, CMU-CS-97-144.

[7] Taylor, R.N. Medvidovic, N. Anderson, K.M. Whitehead Jr., E.J. Robbins, J.E. Nies, K.A. Oreizy, P. and Dubrow, D.L. A Component- and message-Based Architectural Style for GUI Software, *IEEE Transactions on Software Engineering*, June 1996.

[8] Oreizy, P. Issues in runtime modifications of software architectures. Technical report no.35, Department of Information and Computer Science, University of California Irvine, August. 1996.

[9] Kramer, J. and Magee, J. Self organising software architectures. In *Joint Proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96).* Pages 35-38, ACM-Press, 1996.

[10] IBM *Agent Building Environment Developer's Toolkit, Users Guide, Level 6,* http://www.networking.ibm.com

[11] Andersson, J. Reactive Software Architectures with Architectural Agents. Technical report, Linköpings universitet, 1998.

[12] Garlan, D., Monroe, R.T. and Wile, D. ACME: An Architectural Description Interchange Language. *In Proceedings of the Centre for Advanced Studies Conference (CASCON'97)*, IBM/CAS, November 1997.

[13] Filman, R.E. Achieving "ilities", Presented at the OMG-DARPA-MCC Workshop on Compositional Software Architectures, January, 1998, Available at: http://www.objs.com/workshops/ws9801/papers/

[14] Robben, B., Matthijs, F., Joosen, W. Vanhaute, B. and Verbaeten, P., Components for Non-Functional Requirements, Presented at the Third International Workshop on Component-Oriented Programming (in conjunction with ECOOP'98), Brussels, July, 1998.