Evolving Software Architectures: A Position Paper

Sylvia Stuurman
Delft University of Technology

March 5, 1997

1 Introduction

Facts of life like the ever increasing costs of maintenance and evolution of software, show that requirements are not as static and final as they are usually treated. One requirement of probably all software systems should be a certain degree of flexibility with respect to changes in the remainder of the requirements, due to an inaccurate modeling of the environment or to a changing environment. In several systems, changes in software as an answer to changes of the requirements should be applicable during execution. According to Stankovic in [10], on-line change capabilities will especially be needed in the field of real-time and embedded systems.

The recently emerged field of software architecture addresses the design of the overall system structure. In our opinion, design for change should start at this level.

Software architectures are typically described as a composition of high-level connected components ([5]). The term has often been used to indicate structures representing the development view of a system, i.e. the high-level structure of the code (in [11] for instance, the term software architecture is always used with this meaning).

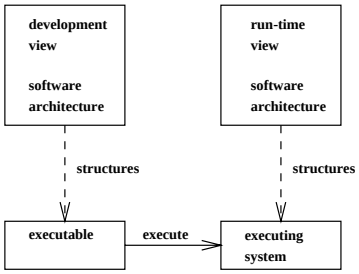

Figure 1: Development and Run-time view Software Architectures

In recent years, software architectures more and more describe the high-level design of the software system as it is seen during execution, with connections representing "interact" relationships as opposed to "implements" relationships ([1]). Figure 1 shows these two usages of software architecture in one figure, where the "run-time view" indicates a software architecture describing a system during execution.

An executing software system usually shows dynamics. In an object-oriented system for instance, objects come and go, and connections between objects may be created dynamically. In a client-server architecture, clients are created and destroyed during execution.

Most of the literature on software architectures addresses the run-time view, but treats software architectures as static structures. This fact is reflected in the languages used to describe software architectures: the dynamics of configurations of components and connections are not captured in most languages ([6]). Exceptions are Darwin ([8]) and Rapide ([7]).

The dynamics of a software architecture are a prerequisite for design for change at the architectural level.

2 Evolvable Software Architectures

The answer to inconsistency caused by changing requirements is a changing system. At the architectural level, such a change means a higher degree of dynamics than the dynamics of a "normally" executing system. Beside changes in the configuration of the architecture, new kinds of components can be introduced.

Idealistically, a change in the software architecture (representing the run-time view) can be applied to the executing system itself in such a way that the system reflects the changed architecture, as is illustrated in figure 2.

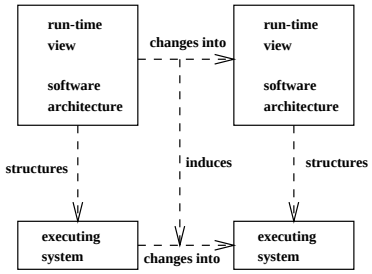

Figure 2: Changing the Software Architecture

We can deduce the following properties for a system with on-line change possibilities at the architectural level:

1. The software architecture is reflected in the executable. Parts of the executable from which components can be instantiated are traceable and replaceable. Obviously, this property is needed to be able to induce a change in the executing system when changing the architecture.

2. Components may be added, deleted or replaced, at execution time.

3. Bindings of components through connections occur dynamically. In other words, connections may be added, deleted or replaced at execution time.

4. Instantiation of components and connections is possible from outside the system.

5. The functionality of components is not directly dependent of other components. Otherwise, adding, deleting or replacing components might not be possible in the running system.

6. It is possible to analyze properties of the system at the architectural level. Before a change is applied, the architecture should be analyzed to guarantee that the changed system will meet the changed requirements.

Every dynamic software architecture shows properties 2 and 3. Properties 1 and 4 are properties of the implementation. Property 5 is a property of the software architecture. Property 6 is a property of the combination of an architecture and an implementation.

3 Independent Functionality of Components

Some descriptions of architectural styles found in literature explicitly state (on-line) change capacities:

- Blackboard architectures support changeability and maintainability because the individual knowledge sources, the control algorithm and the central data structure are strictly separated([4]).

- SPLICE ([3]), a software architecture developed for control systems, consists of independent autonomous processes, each connected to an agent. Agents communicate by message passing, implementing a shared data space. Thus, software functions are isolated from each other. Actual connections between functions are established at run-time, based on names of global data-elements. A system built according to this architecture is dynamically reconfigurable.

- In MESSENGERS ([2]), messages are autonomous objects, deciding at run-time to which node they navigate, and which tasks to perform there. They can carry new function definitions to the nodes they visit. Because messages can be changed or added at run-time, the behaviour of the system can be changed on-line.

- C2 ([9]) is a component- and message based architectural style, designed for applications that have a graphical user interface aspect. A certain degree of independence of components is achieved through "substrate independence": components are structured in a hierarchy, and are unaware of components beneath them. The interface of components (notifications to which a component responds, notifications which a component emits, requests emitted by a component, and requests to which a component responds) are defined independently of the functionality of components.

In the first two examples, direct interdependence of components is prevented by the mechanism of communication through shared data. In the third example, messages are used both to invoke functions of components (nodes), and to change their functionality. As a result, nodes are not directly interdependent, and a mechanism to change the functionality of the system

is available. Components in the fourth example still are interdependent, but the definition of interfaces creates a certain degree of independence.

None of the examples offers a mechanism for on-line analysis of non-functional requirements, such as performance.

4 Conclusion

On-line analysis and change of software systems is needed as an answer to inconsistency due to a changing environment.

In our opinion, the starting point for the support of such systems lies in the field of software architectures. Research should lead to the development of frameworks (styles, description techniques, analysis techniques and implementation support) for systems with on-line change capabilities. Architectural styles for these frameworks should show independence of functionality of the components. In the implemented systems, components and connections can be added, destroyed and replaced dynamically, from outside the system. The effect of changes to the non-functional requirements of the system can be analyzed before a change is applied.

References

[1] R. Allen and D. Garlan. Beyond definition and use: Architectural interconnection. In *Proceedings of the ACM Interface Definition Language Workshop*, January 1994.

[2] L.F. Bic, M. Fukuda, and M.B. Dillencourt. Distributed computing using autonomous objects. *IEEE Computer*, 29(8):55–61, August 1996.

[3] M. Boasson. Control systems software. *IEEE Transactions on Automatic Control*, 38(7):1094–1107, July 1993.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns, Pattern-oriented Software Architecture*. John Wiley and Sons, 1996.

[5] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, December 1994.

[6] D. Garlan and M. Shaw. Charasteristics of higher-level languages for software architecture. Technical Report CMU-CS-94-210, School of Computer Science, Carnegie Mellon University Pittsburgh, 1994.

[7] D.C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transansactions on Software Engineering*, 21(9):717–734, September 1995.

[8] J. Magee and J. Kramer. Dynamic structure in software architectures. In *Fourth SIGSOFT Symposium on the Foundation of Software Engineering*, 1996.

[9] N. Medvidovic, R.N. Taylor, and E.J. Whitehead. Formal modeling of software architectures at multiple levels of abstraction. In *Proceedings og the California Software Symposium*, pages 28–40, 1996.

[10] J.A. Stankovic. Real-time and embedded systems. Group Report of the Real-Time Working Group of the IEEE Technical Committee on Real-Time Systems, at http://www-ccs.cs.umass.edu/sdcr/rt.ps, 1996.

[11] B. Witt, F.T. Baker, and E.W. Merritt. *Software Architecture and Design: Principles, Models and Methods.* Van Nostrand Reinhold, New York, 1994.