# Issues in Dynamic Software Architectures

Jesper Andersson
Department of Information and Computer Science
Linköpings universitet
jesan@ida.liu.se

25th February 2000

**Abstract**

Software architecture is probably one of the best approaches to take non-functional requirements into consideration early on in a development process. Some of these requirements, for instance performance and high availability, are usually implemented as dynamic reconfigurations. Current architecture design methodology does not provide sufficient support for these types of systems. These dynamic changes require new "tools", for specification and assessment, both during modeling and at run-time. A classification of dynamic architecture types is presented and based on this we discuss important issues that must be resolved if development of applications, where conflicting non-functional requirements are partially implemented using dynamic behavior, should be fully supported.

## 1 Introduction

Software development is about delivering software systems that meet some set of user requirements. Functional requirements describe the users' expectations in terms of services that the software should provide. Other, non-functional requirements, specifies the quality of these services and other aspects of the system. There are many different quality requirements. Some are important from a user perspective, e.g. usability, and some from the perspective of the development organization, e.g. maintainability.

Software is used in many different areas and the quality requirements vary from one domain to another. Industrial and large business applications. Command and control systems, and banking systems are two examples of systems where quality is crucial. Compared to a desktop application these systems often must pass high-availability and dependability tests, and score high on performance benchmarks.

A high-availability requirement on a system is typically specified as a maximum downtime for some period of time, e.g. a week or a year. If a system fails this endeavor, the system supplier is liable for damages and any cost for the system owner. Dependability is more difficult to grasp; it covers several areas, such as fault-tolerance and security.

A problem with quality requirements is that these are difficult to trace to individual system components. Compared to the application functionality, which is typically implemented in one or a limited number of components, the implementation of quality requirements are scattered over the system. Another problem is that many quality requirements are competing with each other. For instance, fault-tolerance is to some extent dependent on extra code that check the application state and functionality that, if something goes wrong, can bring the application into a safe-state. This means overhead processing and reduced system performance.

The difficulty of isolating quality requirements to specific components is reflected in existing modeling notations and tools. Qualities are modeled separately, i.e. some special notation and tool is used to design the system behavior with respect to one quality. This means that many conflicts are not addressed during design, these are often discovered and resolved during implementation and testing.

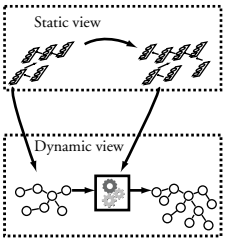


Figure 1: Dynamic Updating Systems

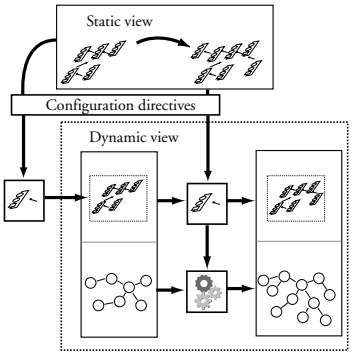


Figure 2: Constructible Dynamism

In industry, dynamic reconfiguration and updating are two techniques often used to achieve high-availability, dependability, and performance. These techniques utilize from dynamic changes to software, but developers have expressed a need for improved tooling, supporting modeling and analysis of the dynamic behavior [1].

Software architecture research, focus on specification and description of software systems at a high-level. A software architecture is a configuration of components and connectors [2]. This perspective is most useful when considering the design of the dynamic aspects of a system, since updating and reconfiguration mainly concerns the manipulation of structures.

Dynamic software architecture is an active area of research within the architecture community. There is, however, still little consensus in the research community on what dynamic architecture really is and when it is used in real-world systems. There are many, similar approaches, to construction, specification, analysis of systems with dynamic architecture.

Even though diversity is a driving force for evolution, we believe that the lack of consensus is a problem. With a defined, stable foundation for dynamic architectures, different approaches can be compared; we can identify possible improvements, and eventually find new interesting combinations. Application developers have also expressed a need for improved support for modeling and validation of applications where dynamic updating and reconfiguration is a central technique in the implementation

In this paper we have surveyed existing approaches to dynamic architecture, and worked out a naive classification scheme, were three generic types of dynamic architecture are described. Based on this classification we identify certain problem areas that needs more research in order to improve support for software developers.

## 2 Dynamic Software Architectures

The capabilities of dynamically modify and update systems is interesting in different application domains. In an historical perspective, dynamic updating has been used in different types of systems, mainly to achieve high availability. A key requirement on a dynamic updating system is that it should keep the application in the same state after a change. Dynamic updating systems are by no means a new field of research. An overview of existing research is presented by Gupta [3].

Figure 1 depicts a dynamic updating system. A system is activated using some initial configuration. At times, new releases of the application are made available. A new release can contain corrected and new components and connectors, some changes in the components require existing connectors to be reconnected to other components. A new release is delivered to the updating system, which introduces the changes to the running system.

Dynamic updating systems are not directly a software architecture problem, and updating systems have

no “architecture-awareness”. But the technique is a fundamental support component for any application with a dynamic architecture

Dynamic architectures, “dynamicism” or “dynamism” are frequently used within the software architecture research community. Luckham and Vera [4], define *dynamicism* in the Rapide language as the capability of modeling architectures in which the number of components, connectors, and bindings may vary when the software system is executed. Medvidovic and Taylor [5], describe *dynamism* as an aspect of configurations. Configurations that allow replication, insertion and removal, and reconnection of architectural elements, statically and dynamically, demonstrate the dynamism property.

The activity of changing architecture configuration dynamically can be divided into four major steps. Initiation of dynamic change, i.e. how the dynamic updates and reconfigurations are initiated. Selection of transformation is the process of selecting an appropriate configuration and how to reconfigure a system. Implementation, i.e. how the actual change is carried out. Assessment of the architecture, i.e. after a change, we need to assess the architecture qualities.

Several ADLs and tools connected to these languages can work with dynamic architectures[5]. Dynamic changes are initiated and carried out externally to the application. Other have extended the capabilities of an architecture framework or middle-ware and added some “intelligence” [6]. The new behaviors replace external activities and take on the responsibility for monitoring applications and dynamically change architectures when needed.

Based on a conscientious study of different approaches to dynamic architectures, we have identified three types of dynamism – constructible, adaptive, and intelligent.

## 2.1 Constructible Dynamism

Constructible dynamism is usually achieved by combining a description language, a modification language and a dynamic updating system. The description language describes the initial configuration of the application architecture. When the architecture should change, changes are described in a modification language. This language typically supports addition and removal of architecture elements, activation and de-activation, and in some systems migrating elements. The dynamic updating can be supported in an architecture framework or some middle-ware.

Constructible dynamism is depicted in figure 2 and described here. The dynamic change is initiated by some event. It can be a monitoring tool that signals a load-balancing problem. A maintainer develops an alternative configuration that resolves the problem. In some situations this can be difficult and the maintainer has to use several iterations where candidates are assessed prior to finding an appropriate candidate. The maintainer implements the changes by sending configuration directives to the application. These directives can act on a first-class representation of the architecture in the application. Some systems take this approach while others have chosen not to have a first-class representation internally. When the changes have been implemented, the maintainer assesses the changes dynamically and verifies the off-line assessment discussed previously.

## 2.2 Adaptive Dynamism

Some applications must have the capability to immediately react to certain events. Here the system can not wait and the initiation, selection and implementation must be integrated into the architecture framework or middle-ware.

Adaptive dynamism is based on a set of predefined configurations that all have been assessed during development. During execution the dynamic changes are initiated by some predefined set of events, the architecture selects an alternative configuration, and implements the reconfiguration. Figure 3 depicts and adaptive architecture where events initiate change. The configurations are kept internally in the architecture

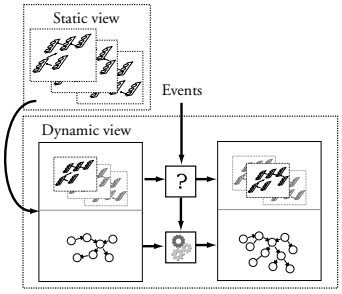


Figure 3: Adaptive Dynamism

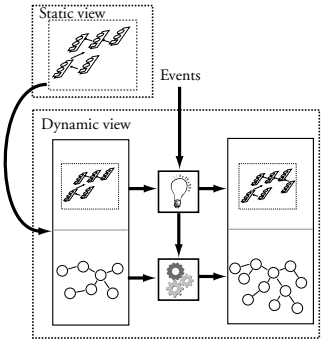


Figure 4: Intelligent Dynamism

using some architecture representation. For every configuration the system needs to know how to proceed if it should change this into some other configuration.

### 2.3 Intelligent Dynamism

The next step, after adaptive dynamism, is to remove the restriction with a limited set of predefined configurations. Figure 4 depicts an application with an intelligent architecture. Compared to the adaptive architectures an intelligent architecture has improved the functionality for selection transformations. The adaptive architectures choose from a fixed set of configurations, an intelligent architecture includes functionality for dynamically construct candidate configurations. The architecture can also assess qualities that the configuration should have. This is a really complicated activity and in practice not many systems use this approach.

## 3 Issues

Dynamic architectures is an important area, especially to large industrial-strength software system, but also in new, emerging software technologies, such as pervasive computing. The architecture and its dynamic properties are important both when developing systems and at run-time, to dynamically maintain applications.

There are many issues that we believe are important for the study of dynamic architectures. Constructible dynamism is more or less nothing else than dynamic updating at an architectural level, so we argue that "static" techniques and tools can be used here. Instead we focus on issues for adaptive and intelligent architectures, which we believe are much more complicated. We claim that modeling of these architectures, needs to be "as close to" run-time as possible, i.e. simulations and other dynamic validation techniques must be used. This means that the tools we need during modeling is similar to the functionality needed at run-time. The issues presented below depend upon access to a suitable first-class representation of architecture configurations in the running application. We also need defined processes that controls the development, such as the method for constructing multi-tolerant systems, presented by Arora and Kulkarni [7]. An approach, which has much in common with our ideas.

*Initiation* – Modeling of dynamic architectures require support for expressing "when" and "why" the architecture should reconfigure. Currently this has limited support in existing modeling languages and we need more expressive notations. At run-time initiation needs support from some event mechanisms in the architecture framework.

*Selection of Transformation* – Dynamically selecting a configuration is an extremely complex operation. Here external components must be used. During modeling this functionality can be used for simulations.

The components must be capable of generating new candidate configurations and perform assessment of these with respect to specific qualities. This area is more or less unexplored.

*Assessment* – Architecture assessment is probably the most exiting problem we see in the future. Reactive systems with an intelligent architecture need automated procedures for assessment of current configurations, candidate architectures, and the transformed architectures. Here we will need a technique that uses a configuration and quantified quality measures as input and automatically assesses an architecture configuration. This is a "blue sky" requirement, but in order to fully support intelligent architectures it is needed.

What we have described above should be regarded as a grand-challenge for software architecture research. Industry faces problems like, "changing the work-load on a node". This is not regarded as a very complicated technical problem, but when several qualities should be combined, more than one developer have expressed an interest in techniques and tools where this can be modeled and not just implemented.

## References


[1] J. Andersson. Dynamic Configuration and Updating - an industrial case study. Under preparation, 2000.

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

[3] D. Gupta. *On-line Software Version Change*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, November 1994.

[4] D. C. Luckham and J. Vera. An Event-Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 12(9):717–734, Sep. 1995.

[5] N. Medvidovic and R. N. Taylor. A Framework for Classifying and Comparing Architecture Description Languages. In *Software Engineering – ESEC/FSE '97*, volume 1301 of *Lecture Notes in Computer Science*, pages 60–76, September 1997.

[6] J. Andersson. Towards Reactive Software Architectures. Technical report, Linköpings universitet, May 1999. Licentiate Thesis. In Linköping Studies in Science and Technology, No. 769.

[7] A. Arora and S.S. Kulkarni. Component Based Design of Multitolerant Systems. *Transactions on Software Engineering*, 24(1):63–78, January 1998.