Dimensions of Dynamism*

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

Introduction

Producing high quality software, on time, and keeping costs within reasonable bounds have been three major goals from the very beginning of software engineering as an engineering science. To achieve this, several theories, methods, and tools have been developed, all supporting some part of the total idealized design process. Even though software developers have a wide range of tools and techniques available today, there are still severe problems. These are most directly associated with the increased complexity inherent in modern software systems. As a result, hardly ever are deployed systems either error-free or fully functionally satisfactory. The result is that, once brought into operation, systems undergo a series of "patches", "fixes", modifications, and changes.

At the same time, increasingly many functions dependent on software in business, industry, and the home are depended on to be immediately available. Having these services unavailable due to updating the software is annoying for users. And there are other types of systems – real-time systems – that cannot afford to be taken down, for instance, telecommunication systems, command-and-control systems, and any other systems requiring continuous operation.

In a not-so-far-distant future, there will be an increasing number of applications with massive distribution and mobility characteristics adding considerably to that class of system which must be modified during operation or "on the fly". Applications will no longer be huge monoliths, on the contrary, the current trend is towards thin clients where functionality is plugged-in on "when–needed–only" basis.

For the systems described above, a partial solution lies in dynamic reconfiguration wherein changes are introduced dynamically and incrementally without affecting the state or considerably reducing the performance of the system.

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 [1, 2]. This perspective is most useful when considering the design of the dynamic aspects of a system, since reconfiguration mainly concerns the manipulation of structures. Systems with dynamic 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, on how to construct, specify, analyze systems with dynamic architectures, and on what initiates dynamic change and where the locus of control is.

In this position paper we present our view on dynamic software architectures by identifying a set of properties or aspects, which can be used to describe the dynamic characteristics of application architectures. These have been discussed by several authors, but with a different terminology. We present a simple scheme were six properties generates three dimensions. Our hypothesis is that all applications with dynamic architecture will fit into this 3-dimensional space.

*This material is based upon work sponsored by the Swedish National Board for Industrial and Technical Development–NUTEK

Within the workshop framework we would like to discuss these ideas and improve them, sharpen our arguments, and identify more attributes that are interesting.

Dimensions of Dynamism

The systematic architectural representations, most easily and clearly expressed as combinations of components and connectors, provides designers and system developers with a logical foundation upon which to make appropriate design decisions, perform testing, and implement system functionality.

Dynamic reconfiguration is best discussed via these components and connectors concepts. It is especially important that early in the design process a common understanding within the design group is reached about this issue in order to properly reflect the consequences and effects in later design and implementation. Consequently, we strongly recommend taking dynamic reconfiguration issues into account at the highest levels of design abstraction.

Support for dynamic architectures has been included in several ADLs and support tools. The term "dynamism" is widely used and refers to different aspects of dynamic architectures. Luckham and Vera [3], 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 [4], 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.

There are some important aspects of dynamic architecture that will have an impact on general architectural issues. First we have the *modeling* capabilities. ADLs must include capabilities to capture dynamically evolving architectures. This means that it must provide means for a designer to express the "what", i.e. *how* it is reconfigured dynamically. This requires ADLs to include capabilities to model different modifications. Medvidovic proposes a "construction notation" [5], that consists of a set of operators used to set up an application architecture or specify run-time changes to architectures. It included operators for creation, deletion, and connection of architectural elements. This type of notation will probably have to expand into a Architecture Modification Language (AML) and provide more complex operators and allow for user-defined operations.

Another important aspect is *run-time.* One important part at run-time is *checking* evolving architectures for different properties, for instance style conformance. Interesting questions here is if restrictions on changes should be included in specifications and if checking is performed before the change or during the change.

At run-time there must be means to reflect changes made to an architecture onto the executing application. Another important run-time aspect is the *state* of the executing application. For instance in a object-oriented system a modification to a component can affect several objects. These objects must be migrated to the new representation in order to preserve the state.

In order to clarify what a dynamic architecture is we need to identify a set of properties that can be used to characterize dynamic architectures. In Figure 1 we display three dimensions of dynamism; *Initiation*, *Type*, and *Control.* Properties from these three dimensions can be combined. For example an Externally initiated–Internally controlled–Updating reconfiguration.

Initiation of Change

Another important aspect of architectural dynamism is when and how the modification tasks are initiated. We see two types of events that initiate reconfigurations, external and internal.

External initiation — External initiation is suitable for planned reconfigurations, for example upgrading parts of applications, migrating to new hardware etc.

Internal initiation — Internal events that initiate reconfigurations can be load-balancing

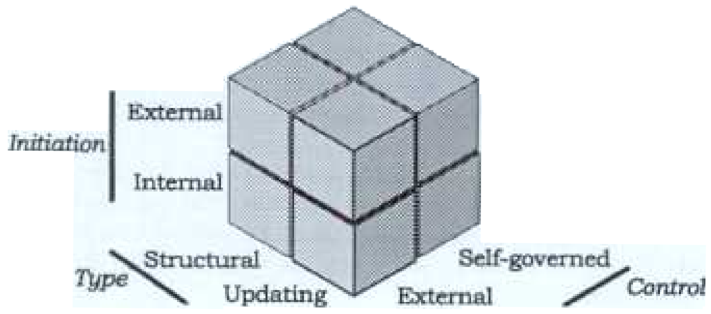

Figure 1: Dimensions of Dynamism

or other exception events in the application or external events that the application detects, such as communication problems and hardware failures.

Type of Change

The type-of-change dimension is important since it is here we can see what can happen to an architecture at run-time. The type of changes that are made to an application will have an impact on other important aspects, such as restriction and checking of run-time changes.

Structural Change — The configuration structure is not preserved when a modification is applied to an architecture , i.e. the set of components, connectors and bindings is affected. To exemplify a structural change, imagine a robust client server application where clients connect to secondary servers when communication to their primary servers fails. Here the architecture changes its configuration when a client opens up a communication line to a secondary server.

Updating Change — The configuration is preserved, i.e. the set of components, the set of connectors, and the set of bindings is not changed. Updating reconfigurations are used when new versions or variants of components are introduced in the system. For instance, imagine a system where a scheduling strategy has been developed and implemented, the system retrieves the new variant of the strategy component and introduces it to the application.

Control of Change

This dimension determines where the locus of control is situated. The control aspect concern both checking of modifications in the pipe-line for inconsitencies and performing the actual modifications.

Externally Controlled Change — External reconfiguration is controlled from the outside. For instance, a system operator opens a dialogue with the application, sending a command sequence controlling the reconfiguration. This approach is simple and straightforward. Other events, which do not require direct action to be taken, can also be handled from the outside.

Self-governed Change — Some reconfiguration tasks, initiated internally, are not easy to predict and plan for and some require direct measures in order to satisfy system requirements. These types need a different approach where the software itself performs architectural reconfiguration without external involvement. For instance, a hardware failure initiates a reconfiguration task. The system must react directly to this event and cannot afford to wait for a manual reconfiguration.

Conclusions

In this position paper we have presented our view on dynamic software architectures. Dynamic architectures require additional support for designers during modeling and for operators at run-time.

In the modeling or specification activity designers need to specify run-time changes to the architecture, i.e. how the architecture will change. Means for specifying constraints on dynamic evolution must also be provided. Some types of architectures also need means for expressing when a specific reconfiguration task should be initiated and how it should be carried out in the specification.

At run-time, support for creating dynamic architectures must be provided. This type of tools or run-time systems include functionality for creation and modification of dynamic architectures, support for external and internal initiation and control, can perform run-time consistency checks etc. Another important part is preserving the state of the application when changing the architecture.

We have discussed a simple scheme where different properties of dynamic architectures create three dimensions. There probably exists more properties. For instance how freely can the architecture evolve. Do we find systems with *Constrained Evolution*, where all changes are known a priori and *Unconstrained Evolution*, where, in principle, all changes are permitted and the system can evolve in an "not-planned-for" manner.

Another important property is *transparency*. In a *Fully Transparent* environment, implementors disregard the fact that the architecture of their application can change dynamically. The opposite end on this dimension is when implementors have full responsibility for initiation and control of the changes.

References

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

[2] D. Garlan and M. Shaw. An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, 1, 1993. World Scientific Publishing.

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

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

[5] N. Medvidovic. ADLs and Dynamic Architecture Changes. In Laura Vidal, Anthony Finkelstein, George Spanoudakis, and Alexander L. Wolf, editors, *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, 1996. SIGSOFT'96.