Exploring approaches to dynamic adaptation

Jorge Fox
Lero - The Irish Software Engineering Research Centre
Distributed Systems Group
School of Computer Science and Statistics
Trinity College Dublin, Ireland

Siobhán Clarke
Lero - The Irish Software Engineering Research Centre
Distributed Systems Group
School of Computer Science and Statistics
Trinity College Dublin, Ireland

ABSTRACT

In this work, we compare current approaches to dynamic adaptation (DA) and identify the need for further research on mechanisms for DA, which should allow for higher compositionality and flexibility. Moreover, after exploring the research landscape in DA we identified the need for a framework that permits to compose several elements of a software system and specially the ones that perform adaptation. Finally, we identified the need for a framework that allows for runtime discovery or replacement of services with a runtime environment capable of verifying the reliability of changes and preservation of the execution time bounds of the software system.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Systems dynamic adaptation, Comparison of languages for dynamic adaptation

1. INTRODUCTION

Dynamic adaptation is gradually becoming a key element in software engineering for a growing range of domains such as: automotive systems, web services, networks, among others. Furthermore, within these domains the requirement to adapt to changing conditions in the environment as well as the need to deploy (additional) services on heterogeneous platforms, motivates the use of technologies facilitating a higher level of adaptation to changes.

A review of the state of the art on Dynamic Adaptation (DA), reveals open research areas. Consider, for instance, time-bounded runtime dynamic systems. As will be explored later in this work, DA within time bounds and without feature interference is a research field in which no conclusive results have been achieved. Naturally, there is a number of approaches for adaptation, but at the same time most are static, see Section 4.1.6. More importantly, the flexibility of adaptation or the degree at which adaptations are achieved, is in most cases limited. Also, in many existing DA frameworks adaptation is achieved by parametrization or reconfiguration, which may render limited solutions with respect to flexibility and limit further adaptations.

The relevance of DA can be demonstrated by the growing need for flexible and dependable systems in complex environments. This means environments characterized by the need for: ubiquity, distributed systems, interoperability; as well as controlled and foreseeable adaptation mechanisms. Likewise, DA is also relevant for ubiquitous systems. Consider the need of software systems to deploy on different execution environments and platforms as [9] mentions:

> Software in the near ubiquitous future (Softure) will need to cope with variability, as software systems get deployed on an increasingly large diversity of computing platforms and operates in different execution environments.

An application area of this technology is in automotive systems. For instance, cars of the future would be in a position to perform self adaptation and take the burden of a myriad possible configurations from the end user.

The remainder of this work is as follows. Section 2 explores the concepts of DA. Section 3 provides an overview on current adaptation techniques and identifies three main types of DA techniques. Section 4 introduces the concepts we use to compare approaches to DA. Next, in Section 5 we outline selected approaches and characterise these in view of the concepts from the previous section. Based on this, we compare some salient properties of current approaches in Section 6. We draw our conclusions in Section 7 and delineate work in progress in Section 8.

2. DYNAMIC ADAPTATION

In this section we explore current concepts and definitions related to DA. First of all, flexibility and adaptability are synonyms in the standard glossary of software engineering terminology [8]. Both are defined as “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.” Second, adaptability is defined as the ability of software systems to withstand changes in their environment. The following quote provides a hint into the nature of adaptation in software systems, mainly at the level of abstraction at which a systems adaptability is enabled: “a software system will be adaptable provided its software architecture is itself adaptable in the first place” [19]. Third, there is a further distinction that needs to be considered, whether we refer to adaptability or adaptiveness. Addressing this distinction is a matter of recognizing the differences between static and dynamic adaptation. As stated in [1]:

> “...Software adaptation can be seen as the ability to reconfigure the software system by the software engineer, hence the term software adaptability, or the ability of the software to reconfigure itself, hence the term software adaptiveness. Software adaptability and adaptiveness are complementary for building conscious design that can accommodate cohesive components built by programmers that are oblivious to the nature of future changes. The intent of software adaptability is to evolve and reuse the software components in future contexts, whereas the intent of software adaptiveness is to enable the software system to alter its behaviour at runtime in order to avoid performance degradation and resource contention.”

In this sense, [6] and [13, 12] introduce a thorough review of adaptability and adaptiveness. In Section 4, we inspired on their classification to develop our comparison framework. However, the emphasis of our work is on dynamic *adaptive* systems.

The notion of adaptive systems we will use in this work is: "The ability to adapt at run time to handle such things as changing user needs, system intrusions or faults, changing operational environment, and resource variability.” (From Dagstuhl Seminar on Software Engineering for Self-Adaptive Systems 13.01.08-18.01.08).

3. OVERVIEW OF ADAPTATION TECHNIQUES

After a review on the literature and an analysis on the current techniques for adaptation, we identified three main groups of adaptation techniques, namely dynamically linking and unlinking selected components, use of generic interceptors and reconfiguration techniques.

Dynamically linking and unlinking selected components.

This technique is used by proposals like iPOJO, where Plain old Java object (POJO) components are binded by handlers added on the base component. These handlers manage on the one hand service publication and providing, and on the other the dependencies. If a service satisfies dependency conditions, then it is published, otherwise it is ignored. Components may turn invalid when a service provider (dependency) is gone. This is the way the iPOJO model handles DA (See [4]). In this type of approaches there is some underlying component model onto which some binding manager is built in order to add new functionality or replace existing components by substituting current communication channels (bindings) by new ones.

Generic interceptors.

The use of generic interceptors is used by approaches like Adaptive CORBA ([17]). These techniques do not modify a component’s behaviour, but intercept the messages between components in order to provide for additional behaviour to perform the adaptation. For instance, in the work of “an adaptive CORBA template” (ACT) generic interceptors are registered with the Object Request Broker (ORB) of a CORBA application at start-up. Interceptors adapt requests, replies and exceptions passing through the ORB. Therefore, the generic interceptors do not modify the component’s behaviour. These interceptors have to be previously registered, which restricts the flexibility of the adaptation. See [12].

Reconfiguration techniques.

These techniques aim at adjusting internal or global parameters in order to respond to changes in the environment. Reconfiguration may help to rearrange the elements of a system. [2] identify two major research approaches to reconfiguration: adding configuration elements and the use of component and configuration languages.

4. A COMPARISON FRAMEWORK FOR DA

In order to classify groups in DA, we first scrutinized the possible lines of research in adaptive systems. This means, the extent to which a system adapts to changes in the environment, whether it is through structural means i. e., architectural adaptation, changes in the parametrization of the system, or a combination of both. Another set of criteria we found, relates to the degree of anticipation to changes. In other words, the extent to which the adaptation reacts to changes in the environment: fully unanticipated or foreseen changes. Clearly, the former is hard to conceive and even more to implement in its pure form. Second, we classify adaptability according to characteristics we identified as relevant for adaptive systems, such as: degree of anticipation, scope of adaptation changes (i. e., architectural vs. localized), whether it is achieved with composition mechanisms or through parametrization and whether there is tool support or not. Equally important, some authors (see [6]) consider the relationship between what is called “compositional” as opposed to “parametric” adaptation, and mixed-forms. We consider both as two dimensions in the classification, which can be combined. Our classification criteria is further explained in Section 4.1. Third, the classification criteria and the approaches we analyzed is represented in Table 1, in which we assigned values (ranging from low to medium and high) to the surveyed research teams for each criteria. Assignment of values was based on a review of the literature and available information. Furthermore, our classification schema draws inspiration from [6], in particular on the distinction on composition adaptation as opposed to parametrical adaptation, and anticipated against unanticipated adaptation.

Table 1 shows the classification criteria we propose for (dynamic) adaptive systems. This graph shows that some criteria can be combined, whilst others may not. Take for instance, achieving adaptation through a high level of parametrization and localized in one or two precise modules, this is relatively straightforward and is present in most approaches. On the other hand, some combinations may not be attainable like having *total anticipation*, meaning full anticipation to environmental changes and achieving it at runtime. Hence, at this stage of our research, results indicate that these criteria are interdependent. Still, it belongs to work in progress to identify the extents and properties of such relationships.

4.1 Classification concepts for DA

We briefly introduce the classification concepts we propose to describe current research approaches in DA.

4.1.1 Unanticipated adaptation

This concept indicates the degree to which the adaptation trig-

Concept/ Approach	ACT (CORBA)	DAiSI	Dynamic TAO	iPOJO	MADAM	MBD DA	PCOM
Unanticipated adaptation	++					+	+
Scope	+	+	++	+	++	+	+
Parametric	+	++	++		++		++
Compositional	++	+	++	++	+	+	+
Dynamic	++	++	++	++		++	++
Static					+	+	
Tools	+	+				+	+

Empty = low level, "+" = medium level, "++" = High level

Table 1: Evaluation of selected research approaches to adaptation

gers and possible adaptation needs are known in advance or not. The higher the level of adaptation to unforeseen changes, the higher the level of the framework in this parameter. We consider that a higher level of adaptation to non foreseeable changes, indicates a more flexible or more generic adaptation framework.

4.1.2 Scope

This concept refers to the extent to which changes in adaptation spread over the software system. We assign values from low to high according to the following. If the adaptation is limited to a localised component, the approach gets the value low in scope, if adaptation is performed on a reduced number of components it is classified as medium level and finally if the adaptation reaches a system-wide level then it is considered high in scope of adaptation.

4.1.3 Parametric adaptation

This criterion indicates whether adaptation is achieved by means of adjusting or fine-tuning predefined parameters in given software entities, such as components, services or methods. A higher parametrization may indicate a rather inflexible framework, due to a higher dependency on predefined values and parameters.

4.1.4 Compositional

This classifier signifies that the framework under analysis achieves adaptations through the insertion or replacement of functional units. By functional units we mean components or sets of components or services. A compositional approach usually relies on binding and unbinding mechanisms.

4.1.5 Dynamic

Reflects whether adaptations may occur after deployment, meaning that the system does not need to halt, yet some conditions for this may be required i. e., such as "quiescence" which means placing a system in a consistent state before and after runtime changes (see [11]). Our proposed criteria "dynamic" helps us to discern those approaches that actually focus on DA and those that rather focus on non-dynamic adaptations.

4.1.6 Static

This category is the opposite of "dynamic," whereas static adaptation means that possible changes are set before deployment or changes require some level of redeployment to be performed. This type of adaptation is usually parametric and the range of circumstances to which the resulting system can anticipate is fixed. This is usually a less expensive solution than the dynamic and unanticipated one, in design and computational terms.

4.1.7 Tools

This classification concept discerns whether the selected approach has tool support, such as a development environment or a runtime monitoring environment.

In the following we introduce the research teams that we considered representative enough to explore our classification criteria. Selection is based on a thorough review of the literature and subsequent selection of teams that had relevant publications in the field. We also privileged those teams working within a consortium of universities and institutions, or an established research group in academia. The objective of our survey is to explain our classification concepts and identify important traits in the field, rather than introducing an exhaustive review of DA approaches.

5. RESEARCH APPROACHES TO DA

In this paper we try to shed some light over the similarities and differences among selected DA approaches. We therefore first provided an overview of adaptation techniques currently used, second we introduced a proposal for a comparison framework for DA and in this section we explore some DA approaches at the light of these comparison framework. The DA technologies selected provide an overview of various methodologies, methods and techniques. Since the purpose of the study was to provide as broad a view as possible on the different technologies available. Our selection was based on choosing research efforts interesting enough to be evaluated against the classification concepts outlined in Section 4.1. Naturally, there are other approaches than the ones we selected. For instance, [12] introduce a taxonomy of compositional adaptation with a broader selection of approaches. In this work we considered some technologies that achieve DA by compositional adaptation, but also technologies that achieve adaptation through reconfiguration or by means of interceptors; these techniques have been introduced in Section 3. For an exhaustive list of adaptive frameworks see [6]. Furthermore, we did not include in this survey approaches that focus on very particular issues like "Hypervisor Modules" [15], or that centre on particular problems of DA such as interoperability [7]. Instead of introducing an exhaustive list of research efforts in DA, we aimed at describing some approaches sharing most characteristics of the current adaptation techniques introduced in Section 3.

5.1 Adaptive CORBA (ACT)

ACT is a language independent template which can be used to develop an object-oriented framework as well as for enhancing -CORBA applications [17]. It introduces generic interceptors, which are specialized request interceptors registered with the ORB at start-up. Interceptors are static or dynamic. Dynamic interceptors can be registered or unregistered at runtime, while static ones cannot be unregistered with the ORB at runtime. This approach also relies on the notion of weaving for relating the dynamic interceptors at runtime. The concept of generic interceptors provides some underpinnings for unanticipated adaptation, since these interceptors are registered without specific behaviour and may later be enhanced at runtime to implement some needed functionality. For these reasons, we positioned it in Table 1 as highly unanticipated, at a middle level scope of adaptation since only dynamic interceptors are changed, middle level of parametrization since the use of proxies and redirection is needed, and highly compositional.

5.2 Dynamic Adaptive System Infrastructure (DAiSI)

This framework focuses on achieving adaptation at the level of component service usage, component service implementation and configuration adaptation. The first kind of adaptation supports switching components at runtime and selecting services based on some quality property, for instance. The second one, supports altering the behaviour of a component and the realization of the service it renders. Finally, the third kind of adaptation is oriented to reconfiguring components in a non-localized way, it aims at modifying how components relate and how the services offered are activated or stopped. For more on it see [10]. It works on the basis of a component model for DA and relies on a formal foundation [16]. Our research indicates that in its current stand DAiSI achieves adaptation through parametrization as well as composition mechanisms. Anticipation to changes seems to be an open issue in this framework, since there is no explicit mechanism to cope with changes and it may not react to unanticipated changes in the environment, rather on those indicated by their configuration component manager (browser). There is a good level of tool support. These characteristics have been summarized in Table 1.

5.3 DynamicTAO and 2*K*

It is an extension to "The ACE ORB" (TAO). TAO is a standard CORBA Object Request Broker (ORB), see [18].

The salient characteristic of DynamicTAO is the capability of reconfiguring the ORB at runtime "by dynamically linking/ unlinking certain components." [18, 14]. It enables remote reconfiguration and replacement of given ORB components with no need to restart the whole ORB, which is a useful trait for DA. It also provides the means for uploading code with new implementations, which is also essential for DA. Given its reconfiguration and replacement capabilities, we marked it in Table 1 as highly dynamic. We also consider that the scope of adaptation, meaning the extent to which the system adapts as proportion of entities with DA capabilities, is in DynamicTAO high, given that the underlying ORB framework allows (at least in principle) for any of the constituent components to be adaptable.

5.4 iPOJO Components

This is a runtime component environment that simplifies development of applications over OSGi, which is a technology focused on facilitating the interoperability of applications and services via a component integration platform [4, 5]. In general terms, iPOJO consists of a component model that "injects" Plain Old Java Objects (POJO's) at runtime. This is the overall mechanism through which systems are adapted in this approach. This is mainly achieved through the management of dependencies and service providing, while the business logic is set at the level of POJO's. DA is then implemented by means of redirecting dependencies; this is managed by handlers which in turn are selected by meta data indicated in XML files. The concept of service used in iPOJO is rather abstract and seems closer to that of features in a broader sense. Moreover, this approach makes the implementation dependant on the underlying service runtime framework, which is why we classify it as having a moderate scope for adaptations. Further in the classification (See Table 1) we identified iPOJO as an approach providing a high level of compositionality; as well as dynamism regarding injection, binding and rebinding of components or POJO's. At the same time, the scope of adaptation is determined by the underlying framework and its availability, which poses limitations to integration with services or components not running on OSGi.

5.5 Mobility and ADaption enAbling Middleware (MADAM)

This framework provides a component model with add-ons for adaptation [6]. With this framework the possible variations for a system are accomplished through the recursive application of predefined realization plans. Realization plans are actual composition plans or predefined combinations of components given by the designer. This component model includes an adaptation manager. A composition or adaptation manager is a common mechanism in most adaptive frameworks. Furthermore, MADAM provides a middleware framework for runtime adaptation with: context management, adaptation management and configuration management. We have therefore classified MADAM in Table 1 as middle level compositional and highly based on parametric adjustments. Likewise, given that adaptations are predefined in an adaptation plan by a designer, we graded it at a low level with respect to unanticipated adaptation.

5.6 Model-Based Development of Dynamically Adaptive Software (MBD DA)

([21, 20]) have worked on reliability aspects of DA. The authors introduce an approach to realise formal models for the behaviour of adaptive programs. This way, they provide a way to ensure that such adaptations are safe with respect to system consistency. It is based on state-machine representations of adaptive programs. The properties that the program should satisfy throughout its execution are called global invariants. Adaptations are defined as adaptation sets and its behaviour is represented as simple adaptive programs. The properties of the adaptive program are local. Their method takes into consideration dependency analyses for target components, specifically determining viable sequences of adaptive actions and those states in which an adaptive action may be applied safely. This technique supports safe adaptation. MBD DA allows for insertion, removal, and replacement of components, in response to changing external conditions. Their work is explored at the example of a wireless multicast video application. In addition a safe DA process has been developed in a related project ([20]).

In relation to our classification, we positioned this approach in Table 1 as static and dynamic. Their state-machine based formal framework does cover static and dynamic analysis. It is also capable of dealing with runtime systems (the axis for "dynamic"). Their approach can be supported by different tool suites (see [20]), so we assigned it a medium level of tool support. There was no stronger evidence of a robust tool set available. This work is more focused on providing a formal framework for analysing adaptation programs than on mechanisms or frameworks supporting adaptation itself.

5.7 A Component System for Pervasive Computing (PCOM)

PCOM is a distributed application model which supports DA via signalling mechanisms and adaptation strategies, see [3]. In PCOM components are entities that interact with each other in order to fulfil their dependencies. This definition of components resembles that of "services," yet services are more explicitly aimed at cooperating, if needed, to fulfil their own functionality. Applications in PCOM are described by a tree of components and their dependencies, being the root component a sort of "`main()`" program or application identifier.

However, it is not clear in [3] whether dependencies only occur following the branches of the tree or some other relationships are allowed and to what extent these dependencies are transitive. Besides that, the authors acknowledge that arbitrary graphs would cause complications. This can be seen as a limitation in the framework. For the above mentioned reasons, we classify it in Table 1 as more parametric than compositional. Given that some strategy for adaptation has to be set beforehand we consider it to represent a medium level of unanticipated adaptation. The framework is not as dynamic as ACT, still does claim to support runtime adaptation, so we considered it highly dynamic as well.

6. COMPARISON OF APPROACHES TO DA

All the approaches we reviewed are bound to a given component model, service model or middleware framework. Our work shows that a generic adaptation model has not yet been achieved.

Our review in Section 5 indicates, first, that most approaches aim at achieving adaptation dynamically i. e., at runtime. This means runtime adaptive systems. At the same time, the implementation mechanism and extent of the dynamicity is variable, e. g. some frameworks may rely on some reconfiguration mechanism (DAiSI), or achieve adaptation by redefining dependencies in the form of a dependency tree (as PCOM). While approaches like DynamicTAO achieve adaptation by dynamically linking and unlinking components as well as providing means to upload code with new implementations. Second, the degree of anticipation to changes differentiates the research groups in a more clear way; most approaches have a fixed set of sources for adaptation and corresponding strategies to cope with them. In this regard, the framework that, in our view, can be considered most adaptive or high in unanticipated adaptation is ACT; which is built for components. Third, the scope of adaptation seems to be related to the level of compositionality and the level of parametrization, the scope of changes is in some cases restricted by the underlying framework, take for instance MADAM a rather parametric approach that does not perform adaptation at runtime. As for the underpinnings of the approaches, not all rely on a formal model as DAiSI or MADAM. Formal methods grant clearer definitions and precision for the framework.

We identified that most frameworks for DA are based on component models. While in the case of service oriented approches, most work in the literature seem to have a broad definition of services that is somewhat same as other definitions such as features. Even more, the distinction between components and services is in most cases not clear, which renders their underpinnings and composition mechanisms unclear. In addition, there is a lack of adaptation mechanisms at the level of services or components logic i. e., behaviour itself. [7]) outlines this in the following lines, we quote:

> Most of the adaptation mechanisms deployed today concentrate typically on content, not so often on communication, but almost never on service logic or behavior itself.

We consider this to be an important issue and guideline for our work in progress.

7. CONCLUSIONS

After a review of a number of DA frameworks and approaches, we detected salient characteristics for DA systems; particularly the extent of changes or what we called the scope of adaptations, whether these are performed at the underlying framework or on a limited number of components pre-enabled for DA. Also, the level of anticipation to changes is an important attribute, because it determines the capacity of the systems to cope with new services or changes in the environment. Moreover, the particular adaptation approaches may vary depending on the underlying foundation: components, services, or a combination of both. However, the adaptation mechanisms themselves are sometimes left to the decision of designers and specified as parameters based on which the system reconfigures or implements the adaptations.

In this work, we identified the need for further research on DA mechanisms; which may allow for higher compositionality and flexibility. Moreover, after exploring the research landscape in DA we identified the need for a framework that manages systems as total functions, i. e., components composed with interchangeable partial functions, services. Finally, there is a need for a framework that allows for runtime discovery or replacement of services, with a runtime environment capable of verifying the reliability of changes and preservation of the execution time bounds of the software system.

8. WORK IN PROGRESS

We continue work to describe the relationships among the classification concepts introduced in Section 4.1, as well as their abstraction levels and their influence on adaptability. We also appraise the limitations for DA, which need to be identified and explained with a formal underpinning. These limitations may include issues such as the need to know the internal or external interfaces of new components or services previous to the adaptation, and achieve some notion of behavioural equivalence that allows us to safely replace components.

ACKNOWLEDGEMENTS

This work was supported in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

9. REFERENCES

[1] Faisal Akkawi, Atef Bader, Daryl Fletcher, Kayed Akkawi, Moussa Ayyash, and Khaled Alzoubi. Software adaptation: A conscious design for oblivious programmers. *Aerospace Conference, 2007 IEEE*, pages 1–12, 3-10 March 2007.

[2] Mehmet Aksit and Zièd Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *ICDCS Workshops*, pages 84–. IEEE Computer Society, 2003.

[3] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - a component system for pervasive computing. *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 67–76, 14-17 March 2004.

[4] Clément Escoffier and Richard S. Hall. Dynamically adaptable applications with iPOJO service components. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2007.

[5] Clément Escoffier, Richard S. Hall, and Philippe Lalanda. iPOJO: an extensible service-oriented component framework. In *IEEE SCC*, pages 474–481. IEEE Computer Society, 2007.

[6] Kurt Geihs. Selbst-adaptive software. *Informatik-Spektrum*, 2007. 0170-6012 (Print) 1432-122X (Online).

[7] Robert Hirschfeld and Katsuya Kawamura. Dynamic service adaptation. *Software - Practice and Experience*, 36(11-12):1115–1131, September/October 2006.

[8] Institute of Electrical and Electronics Engineers. IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries, 1990.

[9] Paola Inverardi. Software of the future is the future of software? volume 4661 NCS, pages 69 – 85, Lucca, Italy, 2007. Software systems;Computing platforms;.

[10] Holger Klus, Dirk Niebuhr, and Andreas Rausch. A component model for dynamic adaptive systems. In Alexander L. Wolf, editor, *Proceedings of the International Workshop on Engineering of software services for pervasive environments (ESSPE 2007)*, pages 21–28, Dubrovnik, Croatia, sep 2007. ACM.

[11] Jeff Kramer and Jeff Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. Software Eng.*, 16(11):1293–1306, 1990.

[12] Philip. K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. A taxonomy of compositional adaptation. Technical Report MSU-CSE-04-17, Dept. Computer Science and Engineering, Michigan State University, 2004.

[13] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[14] Philip. K. McKinley, R. E. Kurt Stirewalt, Betty H. C. Cheng, Laura K. Dillon, and Sandeep Kulkarni. Rapidware: Component-based development of adaptive and dependable middleware. Technical report, Michigan State University, 2005.

[15] Thomas Naughton, Geoffroy Vallée, and Stephen L. Scott. Dynamic adaptation using xen:thoughts and ideas on loadable hypervisor modules. *First Workshop on System-level Virtualization for High Performance Computing (HPCVirt 2007)*, March 2007.

[16] Andreas Rausch. DisCComp – a formal model for distributed concurrent components. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 176(2):5–23, 2007.

[17] S. M. Sadjadi and P. K. McKinley. Act: An adaptive corba template to support unanticipated adaptation. *icdcs*, 00:74–83, 2004.

[18] D C Schmidt, B Natarajan, A Gokhale, N Wang, and C Gill. Tao: A pattern-oriented object request broker for distributed real-time and embedded systems. *IEEE Distributed Systems Online*, 2002.

[19] Narayanan Subramanian. *Adaptable software architecture generation using the nfr approach*. PhD thesis, The University of Texas at Dallas, 2003. Supervisor-Lawrence Chung.

[20] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM.

[21] Ji Zhang, Betty H.C. Cheng, Zhenxiao Yang, and Philip K. McKinley. Enabling safe dynamic component-based software adaptation. *Architecting Dependable Systems III*, 3549/2005:194–211, 2005.