A Survey of Dynamic Adaptation Techniques

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

{firstname.lastname}@cs.tcd.ie

Abstract

The growing complexity of software systems as well as changing conditions in the operating environment demand systems that are more flexible and dependable. A possible solution we envisage is the use of mechanisms for effecting behavioural enhancements or changes in running systems. This has been called Dynamic Adaptation (DA). This implies exploring a number of challenges. Some questions that have to be addressed relate to finding mechanisms for: service detection, implementation of behavioural changes during runtime, service interaction and service behaviour modification. This paper introduces a survey of approaches to dynamic adaptation in order to assess their capabilities. We describe a framework for comparing approaches to (dynamic) adaptation (DA) and evaluate selected approaches to DA against this framework. Based on the comparison framework we outline current trends in DA technologies.

Keywords: Software Engineering, Dynamic Adaptation, Software and Systems Development, Run-time Systems

1 INTRODUCTION

Dynamic Adaptation (DA) 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 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 ones. 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 parametrisation or reconfiguration, which may render limited solutions with respect to flexibility and limit further adaptations.

The relevance of DA lies on the growing need for flexible and dependable systems in complex environments. These are environments characterized by the need for: ubiquity, distributed systems, interoperability; as well as controlled and foreseeable adaptation mechanisms.

2 DYNAMIC ADAPTATION

In this section we explore current concepts and definitions related to DA. Adaptability is defined as the ability of software systems to withstand changes in their environment. As Yan et al. mention “a software system will be adaptable provided its software architecture is itself adaptable in the first place” [18].

Adaptive systems are those that posses the ability to adapt at run-time to react to user needs, system intrusions or faults, changing operational environment, resource, and performance variability. We consider dynamic adaptable systems to be a subset of adaptable systems wrt. the moment of adaptation. Dynamic adaptable systems perform adaptations at run-time as opposed to performing adaptations at design time.

In this sense, [6] and [9, 10] 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.
3 COMPARISON OF APPROACHES

In order to classify groups in DA, we first scrutinised 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 parametrisation 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 foreseeable 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. localised), whether it is achieved with composition mechanisms or through parametrisation 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. Third, the classification criteria and the approaches we analysed 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.

4 CLASSIFICATION CONCEPTS FOR DA

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

4.1 Unanticipated Adaptation

This concept indicates the degree to which the adaptation triggers 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.

4.2 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.

4.3 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 parametrisation may indicate a rather inflexible framework, due to a higher dependency on predefined values and parameters.

4.4 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.

4.5 Tools

We also consider whether the approach has tools to support dynamic adaptable systems, such as a development environment or a runtime monitoring environment. This is the last classification criteria provided in Table 1. We believe this criteria to be of a relatively high importance given the need to facilitate adoption of the approach or framework.

In the following section, 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 ADAPTATION TECHNIQUES

We identify three main groups of adaptation techniques, these are dynamically linking and unlinking selected components, use of generic interceptors and reconfiguration techniques. These techniques were selected after reviewing the literature and analysing the current techniques for adaptation.

The following DA technologies represent an overview of various methodologies, methods and techniques in the field. Naturally, there are other approaches than the ones we selected, however we consider this selection to be sufficient ground for comparisons.

In this work we considered technologies that achieve DA by compositional adaptation, but also technologies that achieve adaptation through reconfiguration or by means of interceptors. 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 “Hyervisor Modules” [12], or that centre on particular problems of DA such as interoperability [7].

5.1 Dynamically Linking and Unlinking Selected Components

This technique is used by the Extensible Service-Oriented Component Framework (iPOJO) [4]. In iPOJO, Plain Old Java Object (POJO) components are “injected” by handlers onto the base component. These handlers manage service publication and providing as well as dependencies. When a service satisfies given dependency conditions, then it is published, otherwise it is ignored. Components relate to each other connecting through these dependencies. Components turn invalid when a service provider (dependency) is gone. Therefore creation or activation of components is equivalent to publicising its dependencies, whilst deactivating a component is achieved by eliminating dependencies. 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 realised 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.

A component container handles all the service-oriented computing aspects and separates them from the business logic which remains in the base component. iPOJO provides a runtime component environment that simplifies development of applications over the platform provided by the Open Services Gateway Gateway initiative (OSGi). OSGi is a technology aimed at facilitating the interoperability of applications and services through a component integration platform [3, 4]. The concept of service used in iPOJO is rather abstract and seems closer to that of features in a broader sense. This approach makes the implementation dependant on the underlying service runtime framework, which renders this work to have a moderate scope for adaptations. iPOJO provides a high level of compositionalnality as well as dynamism regarding injection, binding and rebinding of components or POJO’s. Whilst 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.

Another technique is represented by PCOM. PCOM is a distributed application model which supports DA via signalling mechanisms and adaptation strategies, see [2]. In PCOM components are entities that interact each other in order to fulfill their dependencies. This definition of components resembles that of “services,” yet services are more explicitly aimed at cooperating, if needed, to fulfill 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 [2] 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 may consider PCOM as more parametric than compositional. Given that some strategy for adaptation has to be set beforehand it achieves a medium level of unanticipated adaptation. The framework is not as dynamic as ACT (see Sect. 5.3), still does claim to support runtime adaption, so we considered it highly dynamic as well.

Another group of techniques, closely related to the ones in dynamic linking and unlinking of components propose the use of composition frameworks, filters, paths and injectors. In this category we find a technique that introduces the use of “injectors”. Injectors in The Object Infrastructure Framework (OIF) offer a way to facilitate evolution and creation of distributed systems. Its main mechanism is injecting behaviour on the communication path between components [5]. Behaviours may be injected on the client or the server. Instances and methods can have a distinct sequence of injectors. Stubs can be changed during execution fostering the dynamic behaviour of the system. There is a high-level specification language and a compiler to support...
OIF. OIF injectors work with the Common Object Request Broker Architecture (CORBA) stubs with some modifications on skeletons to obtain the injector sequence for each method. The injector may modify the target, the operation arguments, the annotations, and the return value. It can also invoke other remote calls. Injections may perform actions before and after the server action. This allows to modify the flow of control. In OIF components are black-box objects. Injectors are created by two classes, the injector itself and a factory that creates instances of the injector. Injector instances are created by calls on the factory when building CORBA proxies. Injectors are then inserted in the methods using an aspect-oriented programming language.

Client side injectors can change the destination of a request. There are different kinds of injectors: rebinding, impatient, insecure, mediating, and balancing. These differ on the decision criteria to select target services. In order to determine the target of a redirection, the injector may rely on a “clerk” which possesses information on the alternatives offered by target services. Clerks can be dynamically arranged in case new services are discovered. To optimize this mechanism clerks can be grouped in a “community” of clerks which share information.

### 5.2 Dynamic Adaptation with Aspect-orientation

Dynamic adaptation with aspect orientation (AO) in Yang et al [19] is performed in two phases. In the first phase adaptation points are defined and in the second phase the adaptation infrastructure is related to the base program. As Yang et al say, the adaptation infrastructure consists of an adaptation manager and a rule base. Dynamic adaptation is directed through a set of rules. The adaptation kernel is a loose grouping of adaptation managers that are explicitly invoked to check execution conditions and perform adaptations accordingly. At run-time the adapt-ready program is instantiated. Behaviour adaptors in the running program use a filter-chain to trap the respective adaptation manager and determine which rules are satisfied and what corresponding adaptation should be performed.

### 5.3 Generic Interceptors

The use of generic interceptors is used by approaches like Adaptive CORBA ([16]). 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 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 [9].

ACT is a language independent template which can be used to develop an object-oriented framework as well as for enhancing CORBA applications [16]. It introduces generic interceptors, which are specialised 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 consider this work to achieve a high level of unanticipated adaptation, while achieving only a middle level scope of adaptation since only dynamic interceptors are changed. It also has a middle level of parametrisation since the use of proxies and redirection is needed, and it is highly compositional.

### 5.4 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

<table>
<thead>
<tr>
<th>Concept/Approach</th>
<th>ACT (CORBA)</th>
<th>DAISI</th>
<th>Dynamic</th>
<th>iPOJO</th>
<th>MADAM</th>
<th>MBD DA</th>
<th>PCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unanticipated adaptation</td>
<td>medium</td>
<td></td>
<td></td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Scope</td>
<td>low</td>
<td>low</td>
<td>medium</td>
<td>low</td>
<td>medium</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Parametric</td>
<td>low</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>Compositional</td>
<td>medium</td>
<td>low</td>
<td>medium</td>
<td>medium</td>
<td>low</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Tools</td>
<td>low</td>
<td>low</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Evaluation of selected research approaches to adaptation
of a system. Aksit and Choukair [1] identify two major research approaches to reconfiguration: adding configuration elements and the use of component and configuration languages.

Dynamic reconfiguration [13, 14] aims at 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 realisation of the service it renders. Finally, the third kind of adaptation is oriented to reconfiguring components in a non-localised way, it aims at modifying how components relate and how the services offered are activated or stopped. For more on it see [8]. It works on the basis of a component model for DA and relies on a formal foundation [15]. A related framework is the Dynamic System Infrastructure (DAiSI) [8]. This infrastructure introduces a dynamic adaptive component model which defines how a component has to be structured for DA. Our research indicates that in its current state DAiSI achieves adaptation through parametrisation 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. Another framework is Dynamic TAO, an extension to “The ACE ORB” (TAO). TAO is a standard CORBA Object Request Broker (ORB), see [17]. The salient characteristic of Dynamic TAO is the capability of reconfiguring the ORB at runtime “by dynamically linking/ unlinking certain components.” [11, 17]. 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 consider it to be highly dynamic. We also consider that the scope of adaptation, meaning the extent to which the system adapts as a proportion of entities with DA capabilities, is in Dynamic TAO high, given that the underlying ORB framework allows, at least in principle, for any of the constituent components to be adaptable.

Another framework is 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 realisation plans. Realisation 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. Its composition is based on parametric adjustments. Likewise, given that adaptations are predefined in an adaptation plan by a designer, unanticipated adaptation is not possible.

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

Zhang and H.C. Cheng ([20, 21]) 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 multicasting application. In addition a safe DA process has been developed in a related project [20]).

Their state-machine based formal framework does cover static and dynamic analysis. It is also capable of dealing with runtime systems. Their approach can be supported by different tool suites (see [20]). It offers 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.

6 CONCLUSIONS

After a review of a number of DA frameworks and approaches, we highlighted 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. Another aspect is that the adaptation mechanisms themselves are sometimes left to the decision of designers and are specified as parameters 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. Some traits we recognise as significant for DA systems are the breadth of the adaptation, the mechanisms used to achieve adaptation, and the underlying framework or tools available. Another question that influences DA is run-time discovery and replacement of services, and the decision making process behind adaptation.

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. Specific attributes play a critical role in DA, these are ensuring reliability of the adaptations and preservation of the execution time of the software system after adaptation, and keeping adaptation time within predefined time bounds. In this regards, our survey reveals open areas of research.

ACKNOWLEDGEMENTS

This work was supported in part, by Science Foundation Ireland grant 03/CE2/1303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References