PREDICTABLE DYNAMIC DEPLOYMENT OF COMPONENTS IN EMBEDDED SYSTEMS

Ana Petričić
Mälardalen Research and Technology Centre
Mälardalen University, Västerås, Sweden
ana.petricic@mdh.se
September, 2009
# Table of contents

1  RESEARCH AREA .................................................................................................................. 3  
   1.1  INTRODUCTION ............................................................................................................. 3  
   1.2  MAIN FOCUS - EMBEDDED SYSTEMS ......................................................................... 3  
   1.3  DYNAMIC DEPLOYMENT IN COMPONENT-BASED FRAMEWORKS ............................. 4  
   1.4  DYNAMIC DEPLOYMENT MECHANISM ...................................................................... 4  
   1.5  RESEARCH GOAL ........................................................................................................... 5  
2  RELATED WORK .................................................................................................................... 6  
   2.1  GNU/EDMA PROJECT .................................................................................................. 6  
   2.2  ROBOCOP ..................................................................................................................... 7  
   2.3  OSGi COMPONENT MODEL ......................................................................................... 9  
3  CONFERENCES .................................................................................................................. 11  
4  RESEARCH GROUPS ........................................................................................................... 12  
5  IMPORTANT LITERATURE .................................................................................................. 13  
6  RESEARCH METHODOLOGY .............................................................................................. 13  
7  REFERENCES ....................................................................................................................... 14
1 Research area

1.1 Introduction

The domain of my research is component-based software engineering (CBSE). This is an emerging approach in software engineering which tends to build software systems out of pre-existing, and reusable components. Components are thus developed and tested independently and separate from systems they compose.

One of advanced features of component-based systems is the possibility to update new or replace old components in order to improve the system characteristics. This characteristic has proved to be very beneficial as today’s software evolves over time and improvements in software are developed in rapid succession. In consumer electronics (CE) for example, the value and the economic life-time of a device and the software on it can be increased by upgrading and extension of the software.

There are particular cases where applications require a very high level of availability in the sense that they cannot be stopped. Therefore operations like deployment of new versions, patches or extended features need to be done while the application is running. This process is usually referred as *dynamic deployment*\(^1\). Examples are most mission critical areas such as telecommunication, banking systems, avionics, and industry control. Interruption of these systems implies high cost (think of shutting down one industry pipeline) and even life-threatening (think of shutting down one life-support system). Consequently, maintaining and upgrading of a system while preserving continuous availability of services is an issue of increasing importance.

1.2 Main focus - embedded systems

In my work, I will focus on dynamic deployment mechanisms for embedded systems. Embedded systems span all aspects of modern life and most, if not all, of the system examples mentioned above consist of embedded systems. They range from portable devices such as cell-phones and MP3 players (consumer electronics), to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity of embedded systems varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

Embedded devices are characterized by their long life cycle, and as such require and can always benefit from seamless upgrading of their software. Except for upgrading, dynamic software evolution allows the system to adapt its functionality in response to frequent changes in the device’s context. Finally, devices can be customized to the needs of a consumer in the period that it is owned and used by the consumer.

As they are dedicated to specific tasks, embedded systems are always optimized. Some have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low performance requirements, allowing the system hardware to be simplified to reduce costs. They run with limited computer hardware resources, in most cases little memory. All these characteristics imply many constraints imposed at embedded systems, which makes system updating or upgrading, a very delicate operation.

---

\(^1\) From perspective of a component we use term *dynamic deployment*. When talking from system’s point of view, we use term *dynamic reconfiguration* (refers to changing, updating or otherwise modifying system during execution).
Due to these constraints, except for functional verification, in case of upgrading an embedded system, detailed performance verification has to be done, in order to be sure the system will keep its characteristics after new component has been deployed. If we speak about dynamic deployment, then this task is even harder, as the target system has to keep running and cannot be used for testing. There are cases where system replica exists, but also there are cases such as telecommunication domain where it is extremely hard to simulate the system or to make its replica. In such a case it is necessary to have mechanisms to specify system requirements constraints, as well as characteristics of each component, and to provide ways for testing new component in order to confirm its compliance with the target system.

1.3 Dynamic deployment in component-based frameworks

The problem of dynamic reconfiguration is not new. It is almost as old as software engineering itself, but it especially became apparent with development and expansion of distributed systems. One of the first works on this topic was the one by Kramer and Magee [14].

Component based software engineering and component based frameworks provide an infrastructure which is suitable for dynamic deployment processes in the sense that only some components of the system have to be updated, changed or deployed instead of the whole application.

Today’s component frameworks can support upgrading and extension at different stages of the software life-cycle (design-time, compile-time, run-time, etc.). Several general-purpose component technologies support dynamic deployment of components, such as OSGi framework [20], Robocup [19, 23, 16] or SOFA [21] (first two are explained below in more detail). Run-time upgrading is also supported by some industrial component models like COM [2], EJB [24, 26] and .NET [15]. However, the requirements on a specific component framework highly depend on the problem domain in which the framework will be used. For this reason, the feature of dynamic deployment often has problems with system dependability; a new component can break the system’s integrity, which can lead to the system failure or degradation of its performance. The mechanisms that enable dynamic upload require certain systems resources which are not critical for large systems, but are not under the control of the system. As such, these mechanisms are not appropriate for embedded systems.

Another, even important issue is the validation. Most of the existing solutions focus on the upgrading mechanism, and only a few are mentioning the validation of deployed components. This is currently a hot issue for embedded systems due to the constraints imposed upon them.

1.4 Dynamic deployment mechanism

In the context of run-time reconfiguration, a distinction is usually made between functional and structural changes. Functional changes include the addition of new/improved code to a running system in order to modify/upgrade its functionality, whereas structural changes refer to reconfigurations that change the relationship between different components of the system, or replicate portions of the application for execution on a different machine. The focus of my research will be on functional changes.

Most component-based approaches consider contractually defined components with behaviour specifying interfaces for checking consistency and interoperability. When updating a component, there are several possible cases that affect the deployment process:

- Changes are not affecting external interface of the component i.e. some methods were being re-implemented and improved.
• Changes to the component are affecting its external interface in a sense that the update only needs to preserve the old interface being possible to add new methods to it in order for new client to use them.

• Some approaches such as GNU/EDMA [18] consider cases where external interface of the component is being changed. One of the solutions is to insert new, adaptor component between the current clients and the new updated component. By doing so, all method invocations pass through the adaptor which performs appropriate transformation of parameters to enable execution process.

The process of dynamic deployment usually consists of the following four steps (the mechanism needed for uploading new functionality can be largely shared with that for upgrading):

• Initiation of a change
• Identification of affected components (identification of change type and scope)
• Accomplishment of the upgrading
• Analysis/check of the consistency

Various existing approaches have different focus on a particular step and thus aiming at different goals. However, most of them, are lacking the last phase, either completely (e.g. GNU/EDMA described below in more detail), or it is predicted in the model but a little work is done on that account (e.g. Robocop).

The problem of validation raises many questions:

• How to inspect system/component's requirements and performance attributes?
  ▪ Are these are obtained by inspection/introspection?
• How to define system requirements (functional and non-functional) in the system model? The same question arises for system components as well for new components that have to be deployed to the system.
• How to keep specified requirements during run-time? Are they kept in the system?
• What aspects have to be tested in order to be sure that the new component is suitable for target system?
• Since the target system has to keep running (i.e. it is unavailable), how to run the validation?
• What are the entities that take part in dynamic deployment mechanism? What entities are in charge of performing the validation?
• How to ensure low resource consumption (e.g. memory consumption) of deployment mechanism and minimized system response?

These are the questions that I will focus on in my research.

1.5 Research goal

The goal of my research is to develop a technology for dynamic deployment of components in embedded systems domain that meets the following requirements:

• A resource efficient mechanism for dynamic component deployment. The mechanism can be adopted from existing component models or service-oriented architectures, and adapted for usage in embedded systems domain.
• Predictable deployment. This includes procedure of verification whether a component to be deployed satisfies system requirements and whether a system can provide conditions defined by component’s requirements.

2 Related work
There is a lot of work in the area of run-time updating and deployment of components, and it has been an active research field especially in last few years. But very few of them are focusing on embedded systems domain.

In the beginning the problem of component deployment and adaptation was mainly tackled from programmer’s point of view, with proposals that aim at easing reuse of existing components in the development of new applications. Most proposals in this category were based on code modification [3, 1, 12, 13]. There were proposals on adapting running applications, based on wrappers [21] or meta-level architectures [17].

Many approaches focus on updating of procedural systems (e.g. [7, 9, 10]). Using indirection, a method call is diverted to a new/updated version of the procedure. Wrappers are used to handle changes to interface and in some cases state is transferred using mapping functions. Dynamic updates can be initiated by the application itself or by an external event. However, applying procedure-based approaches to component-oriented applications is extremely difficult, since this breaks component-oriented abstractions and type safety. Therefore, these systems will not be considered any further in my research.

A simple method for implementing dynamic deployment in an object-oriented system is by using the Strategy Design Pattern [8]. This solution allows for changing an algorithm or strategy at run-time, but it requires that the application is designed with future changes in mind, and that all possible strategies must be known at compile-time. Therefore, it is not suitable for unanticipated updating which is in the focus of my work.

Kramer and Magee had the first work on evolution of component-based systems. In [14] they describe a system in which a programmer can specify the desired changes in a declarative way. For safe removal of a component from the running system they introduce the concept of quiescent nodes. They did not attempt to transfer state between different versions of a component, so their system needs extension in order to truly implement dynamic updating.

A very flexible approach can be found in meta-level architectures (e.g. [5, 22]). Through reification of object oriented concepts (class, method-call etc.), a meta-model is build on top of the application. This way, changes to the application are possible by changing the metamodel. A great advantage of this approach is that this provides a clean separation between the application code and the code responsible for the reconfiguration (meta-code). In some cases it is possible to build an adaptation framework [6], however, it requires the large overhead which is not acceptable in an embedded system context.

In the rest of the section some approaches that support dynamic deployment in component-based systems are described.

2.1 GNU/EDMA project
GNU EDMA is a modular and open development environment developed by GPI (Grupo de Procesado de Imagen or Image Processing Group) at Vigo University, Spain. It is similar to the component object model or the system object model, and allows using most of the classic OO features (inheritance, virtual method override etc.) dynamically. The main goal of the
The project is to provide ways for building modular and evolving applications as well as highly reusable components.

Main features of GNU/EDMA are:

- Mixing component-based and object-oriented approaches in a way that components can be managed as classes (as in other OO programming languages).
- All common object-oriented features can be done dynamically. For example object inheritance hierarchy can be changed dynamically thus enabling building of dynamically reconfigurable systems.
- Clear separation between analysis/design and implementation stages. This way component functionality can be implemented even with languages that do not support OO paradigm or can use OO features that are not supported by the used language (e.g. using multiple inheritance in Java).
- It is fully extendable. There are built-in subsystems to support extension of internal GNU/EDMA functionality.

GNU/EDMA updating mechanism adds up to copying a new shared library of the component and interface description file to a specific directory. After copying, the component is marked as changed and reloaded by the system next time it is being used. This simple method allows system to keep running while the component is being updated. There is an important restriction on this mechanism that application thread should not be running any of the classes being updated or their super-classes (in case they are being inherited).

GNU/EDMA also deals with some other situations in updating process that can cause broken clients such as virtual method change, property change or standard class refactoring techniques, all three concerning the class being updated and all super-classes on its inheritance hierarchy.

In general case, dynamic deployment process uses the rule file that contains information about arbitrary changes in a form of commands concerning changes that should be made. Rule file is generated by the same refactoring tool used to build the update being installed and is parsed during the update process.

Considering memory consumption and performance of GNU/EDMA, this approach is not appropriate for embedded system domain. For each component instance that is being updated, a new copy of the instance must be made. This means that, the more components involve in the hierarchy to be updated, the more memory is required. Overall response time of the system is minimized by maintaining old versions of components in the system so the clients can still continue its execution and new clients will use new component update.

GNU/EDMA provides a flexible run-time updating system but it misses the verification of component functionality and non-functional requirements in order to avoid the possibility of breaking system integrity.

## 2.2 ROBOCOP

Robocop is a project that defines an open middleware layer for high volume consumer electronics. It aims to support definition, modeling and trading of software components, their use in consumer electronics (CE) applications, and run-time upgrades and reconfiguration of such applications. Most important requirements for CE domain that Robocop deals with are robust and reliable operation, run-time upgrading and extension, low resource footprint and support for component trading.
Robocop defines components as units of trade. A Robocop component is a set of different models that are related to each other. These models address different aspects of the component that can be of interest to different stakeholders. Some examples of such models are interface definition (Robocop IDL), behavioral models, resource consumption models, etc. Robocop is open in the way that it does not limit the number or types of models that a component consists of. A special type of model is the executable component. Executable component is a binary representation of the component that implements its functionality and can be executed. A Robocop component can contain multiple executable components (implementing the same IDL) that are targeted for different platforms or operating systems.

Robocop architecture defines three frameworks that support different concerns of component's life cycle:

- **The Development framework** defines how different stakeholders (e.g. component vendors, system integrators) relate to each other, to the component model and entities like component repositories, and target devices.

- **The Run-time framework** defines a partial architecture for Robocop devices. It consists of the Robocop component model and the Robocop Runtime Environment (RRE). To achieve a minimal resource footprint, standard RRE supports only registration of components and services, and location and instantiation of services. Robocop component model defines a standardized part of service interface that, together with RRE, provides mechanisms for run-time binding of components and reconfiguration of Robocop systems.

- **The Download framework** enables run-time upgrades and extension of applications built with Robocop technology. This is achieved by providing mechanisms for locating new components, testing if they are suitable for use in a given system, and transfer of new components from repositories to target devices.

Within Robocop project, there is just a little work done considering tailoring of components and their automated registration within RRE. Considering locating of components and target loading there is a solution proposed which consists of the following phases:

- Locating of entities that participate in the download process
- Decision about feasibility of the download
- Transfer of Robocop component to the device (target loading)
- Confirmation of download and registration of the downloaded component at RRE

In this process there are several roles existing, initiator, locator, decider, repository and target device. These roles can run at different devices, except the target role which runs at the device to which a component is transferred.

Initiator is responsible for verifying that all entities required in the download process are present, and also to coordinate the process of download. Initiator is triggered either by the component upgrading process or as result of a change in user preferences settings.

Locator deals with locating last three entities in the download process, repository which contains the component to be downloaded, target device where component will be deployed and the decider which makes a decision whether the download will take place or not. Locator provides addresses of these entities by maintaining tables of all registered repositories, targets and deciders.

Decider role is here to compare component existing in the repository to the requirements in the target device, and then to decide whether the component is suitable for downloading.
Decision procedure performed by decider may be sophisticated and can depend on the domain, for example in case of real-time systems a schedulability test can be performed. However, until now there is little work been done on deciding techniques in Robocop.

The download procedure is shown of figure below.

![Figure 1. The Robocop download procedure](image)

Considering resource consumption during dynamic reconfiguration process, Robocop implements resource management through the Resource management framework. The aim of this framework is to prevent resource overloads on embedded devices that support dynamic updates or upgrades. It introduces a notion of resource-aware consumers, which are application entities that have information about resources needed for its functioning. Special types of such entities are the quality-aware consumers, which consume different amount of resources depending on the level of quality they provide in a given moment. The consumers can register their resource needs to the framework, which can then guarantee them requested resources or deny their request. The framework can also optimize system quality depending on the available resources. A solution of memory consumption of Robocop applications is given in [11].

### 2.3 OSGi component model

The OSGi stands for Open Service Gateway initiative, an industry alliance founded in 1999, with a mission to specify and develop an open service platform for delivery and management of multiple applications and services to all types of network devices in home, vehicles, telecommunication network and many other environments. The OSGi emphasis is on a lightweight framework that can be executed in low-memory devices such as all kinds of embedded systems. To ensure portability on different hardware, OSGi relies on Java. The important characteristic of OSGi is that it supports dynamic evolution of a system. Components can be downloaded, updated, and removed dynamically, without even stopping the system. Moreover, the OSGI allows for remote administration of the system via the network.
OSGi is based on two main concepts that can both be interpreted as components, *bundles* and *services*. Application is developed as a set of bundles that contain services. Each service (defined as an implementation and as a set of interfaces) implements a part of the overall functionality. A service can be considered as a unit of composition; and a system as a set of cooperating services that are connected. On the other hand, bundle is a unit of deployment that groups a set of services that can be dynamically deployed as a unit.

![Figure 2. OSGi bundle component interface](image)

From external point of view, a bundle can be presented as shown in Figure 2 (figure is taken from [4]). We can distinguish three kinds of interaction points:

- Interaction with traditional technology. A bundle may require and provide one or more Java packages. This is part of the interface is declared statically (colored in white on the figure).
- Interaction with other bundle components. This part of interface is intended to manage dynamic connections between services, so that service interfaces (either required or provided) can be attached or detached dynamically (colored grey on the figure).
- Interaction with the run-time environment. This is intended for a bundle to listen to events published by the run-time framework such as the insertion of a new component in a system or the publication of a new service. This way, a bundle can adapt to the evolving architecture (colored black in the figure).

From internal point of view, a bundle is represented as a JAR archive containing service components, Java packages, and other files such as configuration files and images. The entities contained in the bundle are not visible from outside as long as the bundle does not export them.

In OSGi, a system is an evolving set of bundle components. Components can connect to each other dynamically based on their own decisions, however this also implies that components cannot assume that the interfaces they use will be available at all times. This means that components may have some knowledge about how to connect and disconnect. Dynamic connection mechanism goes as follows:

- When a bundle component publishes a service interface, it can attach to it a set of properties describing its characteristics.
- When a component requires an interface, it will select one using a query expression based on these properties.
• The final connection is never given explicitly as the result of the query depends on the actual state of the system (depending on the components existing in the system and services they provide at the moment).

• Once a connection is established, there is no guarantee that the service will remain available. Each bundle component must listen to events generated by the OSGI runtime environment and must take appropriate action as the system evolves.

The OSGi framework includes a service gateway, a central entity to enable, consolidate and manage service requests and service delivery between some local area network and wide area network such as the Internet. The framework also provides a hosting environment with the following services:

• Managing life-cycle of bundles
• Resolving interdependencies between bundles and making classes and resources available from a bundle
• Maintaining a registry of services
• Notifying listeners on events when some bundle's state change, when a service is registered or unregistered, or when some error occurs

In the process of dynamic reconfiguration in OSGi, an explicit verification process is missing. However, there are several mechanisms that can ensure compatibility such as the compatibility of the version numbers of requested component interface(s) or already mentioned properties attached to each provided service. On the other hand, comprehensive verification of performance characteristics that is very significant in embedded systems is not supported.

3 Conferences
In my work I will mainly focus on software engineering conferences and workshops that deal with component-based software engineering.

CBSE (The International Symposium on Component Based Software Engineering)
The conference deals with component specification, composition, analysis, verification, industrial experience as well as empirical studies in component-based software engineering. The CBSE symposium has a track record of bringing together researchers and practitioners from a variety of disciplines to promote a better understanding of CBSE from a diversity of perspectives, and to engage in active discussion and debate.

WCOP (The International Workshop on Component-Oriented Programming)
WCOP seeks position papers in the field of component-oriented programming. Further, it solicit papers reporting on experience with component-oriented software systems in practice, where the emphasis is on interesting lessons learned, whether the actual project was a success or a failure.

EUROMICRO SEAA (EUROMICRO Conference on Software Engineering and Advanced Applications)
The conference is bringing together people from business, industry, research, and academia who are working in software engineering and information technology. The conference runs in several tracks, one of them focusing on service and component-based software engineering.
ICSE (The International Conference on Software Engineering)
This is the premier software engineering conference, providing a forum for researchers, practitioners and educators to present and discuss the most recent innovations, trends, experiences, and concerns in the field of software engineering. Every year the conference has a copious program of events, including keynote talks by leaders in the field, invited presentations along specialized themes, tutorials, workshops, and presentations of technical papers on innovative research, the cutting edge of practice, and new developments in software engineering education.

ESEC (The European Software Engineering Conference)
ESEC is an internationally renowned forum for researchers, practitioners, and educators to present and discuss the most recent innovations, trends, experiences, and challenges in the field of software engineering. It includes a variety of events such as challenge presentations, demonstrations, posters, workshops, and a doctoral symposium.

COMPSAC (The International Computer Software and Applications Conference)
This is a major international forum for researchers, practitioners, managers, and policy makers interested in computer software and applications. It gathers researchers to discuss the state of art, new advances, and future trends in software technologies and practices. The conference often includes a number of workshops on emerging important topics. Workshop that was held in the year 2008, was very interesting for my research, it was the workshop on component-based design of resource-constrained systems.

SIPEW (SPEC International Performance Evaluation Workshop)
The goal of the workshop is to bridge the gap between theory and practice in the field of system performance evaluation. This seems to be very interesting workshop for my research as there are topics such as performance modeling and analysis, workload characterization and experimental performance evaluation and performance-oriented design and development.

4 Research groups
Safe and Comprehensible software Components project group
Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia in Pilsen, Czech Republic, group leader: Premek Brada
Research focuses of this group are software components and their modeling, substitutability and versioning. They concentrate on practical use of subtype-based substitutability of components, at present focusing especially on the OSGi platform. Second area is application of extra-functional properties as part of component specification and use.

Methods and models for consistency verification of advanced component-based applications
Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia in Pilsen, Czech Republic
Distributed Systems Research Group, Department of Software Engineering Faculty of Mathematics and Physics, Charles University
This is the project lead in cooperation of two universities (Charles University and University of West Bohemia) which has a goal to create advanced component models and methods that would enable building complex component applications and ensuring their consistency, and to validate their suitability for practical use.

**Embedded and Ubiquitous Systems group**

Part of the Distributed Systems and Computer Networks research group, Department of Computer Science, Katholieke Universiteit Leuven, Belgium

Main topics investigated by this group are runtime adaptability of ubiquitous systems software (target systems are embedded systems), run-time evolution of software in component based systems, model driven development of software for embedded systems, methodologies and notations to describe adaptable software, middleware to support adaptability etc.

**Software Engineering Group**

*University of Oldenburg, Germany*

At this department there is a small group of researchers focusing on runtime reconfiguration of component-based systems, with aims at minimizing the interference caused by the reconfiguration and thus maximizing system responsiveness during reconfiguration.

**Information Systems and Software Engineering (ISSI) research group**

*Department of Information Systems and Computation, Universidad Politécnica de Valencia, Spain*

Among other research lines, they are working on an approach for dynamic reconfiguration of software architectures by taking advantage of aspect-oriented techniques. Their approach enables complex components to autonomously reconfigure themselves (they are capable of both having knowledge of their current configuration and reconfiguring themselves at run-time). This approach has been developed for the PRISMA aspect-oriented architectural model.

**5 Important literature**

Some excellent surveys on dynamic reconfiguration are given by Gupta [9], Hicks [10] and Segal and Frieder [25].

The paper that can be considered as a seminal paper in the area of dynamic reconfiguration is the one by Kramer and Magee [14] where they manage a reconfiguration as a transaction.

**6 Research methodology**

My work has started with a study of different component models and technologies.

I plan to study component models and technologies that support dynamic deployment, and system predictability in respect to different quality attributes. I also plan to make a research on the approaches for dealing with dynamic deployment in service-oriented architectures where dynamic deployment is already in use.

Next step will be to decide on the component model which will be the basis for my work. If it will be possible I will adopt some of existing dynamic reconfiguration approaches (either from component-based or service oriented-architectures), and try to customize it for embedded systems domain. After I have established the basis (appropriate component model and
deployment mechanism), the appropriate verification mechanism has to be defined. This also includes detection of most important non-functional characteristics of embedded systems that need to be verified.

In parallel a work on implementation of a component model for embedded systems (PROGRESS component model) will be done.

7 References


[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Massachusetts, 1994.


