Proceedings of

MOMPES’04

1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software

June 15, 2004, Hamilton, Ontario, Canada

Editors:

João M. Fernandes
Johan Lilius
Ricardo J. Machado
Ivan Porres
Introduction


The Object Management Group’s Model Driven Architecture (MDA) paradigm is an approach to the development of software, based on the separation between the specification of the systems and their implementation using specific platforms. This workshop focuses on the scientific and practical aspects related with the adoption of Model Driven Development (MDD) methodologies (notation, process, methods, and tools) for supporting the construction of pervasive and embedded software.

Suggested areas of interest in the workshop include, but are not restricted to:

- Specification of Platform Independent Models (PIMs) and Platform Specific Models (PSMs)
- PIM to PSM transformations
- MDD process for embedded and pervasive software
- Automatic code generation in MDD contexts
- Testing and validation in MDD contexts
- Tools for MDD of embedded and pervasive software
- Case studies on the application of MDD

We would like to thank the ACSD organisers for help in organising this workshop, and TUCS and CREST for support in publishing the proceedings.

We hope you will enjoy the workshop!

The organisers,

João M. Fernandes, Univ. Minho (PT)
Johan Lilius, TUCS (FI)
Ricardo J. Machado, Univ. Minho (PT)
Ivan Porres, TUCS (FI)
Contents

Javier Muñoz, Vicente Pelechano and Joan Fons, ......................... 3
Model Driven Development of Pervasive Systems

José E. Fernandes, Ricardo J. Machado and João A. Carvalho, ............. 15
Model-Driven Methodologies for Pervasive Information Systems
Development

Marcus Alanen, Johan Liljus, Ivan Porres and Dragos Truscan, ............. 25
Model Driven Engineering: A Position Paper

Yanguo (Michael) Liu and Issa Traore, ............................... 31
UML-based Security Measures of Software Products

Ian Oliver and Juha Savolainen, ...................................... 45
Modelling Message Based Real-Time Systems with UML and
Rate Monotonic Analysis

António J. Esteves and Alberto J. Proença, ............................ 57
A Partition Methodology to Develop Data Flow Dominated
Embedded Systems

Jens Peter Holmegaard, Anders Peter Ravn and Peter Koch, ................. 81
An Approach to Quality Estimation in Model-Based Development

Andreas Bauer and Jan Romberg, ................................. 93
Model-Based Deployment in Automotive Embedded Software:
From a High-Level View to Low-Level Implementations

Manuel Bernardo Barbosa and João Miguel Fernandes, .................. 107
A Model-Based Approach to the Development of Distributed
Control Systems
Abstract

The rise of the number and complexity of pervasive systems is a fact. Pervasive system developers need advanced development methods in order to build better systems in an easy way. The Object Management Group (OMG) promotes in its Model Driven Architecture (MDA) proposal the use of models to describe software systems at a high level of abstraction. MDA provides an appropriate context for building and transforming conceptual models in order to achieve the automatic code generation for software production. In this paper we present a MDA based method for pervasive systems development. Our method introduces concrete techniques for every MDA building block. We introduce PERV-ML, a language for describing pervasive systems in a platform independent way, a set of techniques for building platform specific models for the OSGI framework, and a set of techniques for building model transformations between platform independent models and platform specific models.

1 Introduction

The rise of the number and complexity of pervasive systems is a fact. Computing based systems growth is arriving to all environments of our daily life. This situation requires solid engineering methods in order to develop robust systems. For many years, software engineers have used conceptual models for providing structured and high level views of software systems which are used as a guide for the coding task. Current trends propose that models must play a more important role. These approaches claim that models must be the most important artifacts in the development cycle, therefore systems developers build and transform systems models in order to achieve automatic code generation. The Object Management Group (OMG) has reflected these trends in its Model Driven Architecture (MDA) [8] proposal, that is being adopted as a new industrial strategy. MDA guidelines application to pervasive systems development can help to build better systems in an easy way.
Currently, modeling methods for real-time and embedded systems are being applied to pervasive systems development [10, 5, 9]. These techniques provide to the developer low-level abstraction constructs that directly represent hardware entities. Following this approach system description has a strong dependence on the hardware system. Moreover, any change in the system requirements usually affects to a wide portion of the system model. Another weak point of these approaches is the lack of well defined and automated transformations from system models to implementation. This feature is mandatory in order to provide a true MDA-based development method. Finally, most of these techniques assume that developers are allowed to program system devices which is a very strong assumption in current pervasive systems.

In this paper we present a MDA based method for pervasive systems development. The structure of the paper is the following: Section 2 describes a pervasive system as an integrator system and presents a case of study. Section 3 introduces concrete techniques for each MDA building block in order to develop pervasive systems. Section 4 proposes PERV-ML, a language for describing pervasive systems in a platform independent way. Section 5 outlines the proposed techniques for building platform specific models and for defining model transformations between platform independent models and platform specific models. Finally, section 6 includes some conclusions and further work.

2 A Pervasive System, an Integrator System

Requirements for current and future pervasive systems involve a great diversity of types of services [12]. Such different services as multimedia, communication or automation services need hardware devices that different manufacturers provide. These devices live in several networks running on different technologic platforms, but they can not satisfy all system requirements on. Devices must work together for achieving some system goals. Therefore we can distinguish two sources of service providers: commercial off-the-shelf (COTS) devices 1 and the software system that integrates all the devices of the system.

Our approach considers that the development of a pervasive system consists in the selection of the suitable COTS devices and the development of the software system that integrates them in order to provide the services that users require. Therefore, we can clearly separate a requirements engineering phase where system analysts identify and describe the services that system should provide, and a design phase where system architects specify which devices must implement the services.

A pervasive system for a meeting room can be used to illustrate our approach. In such a system, users require services like lighting management by rooms presence, binds management or draws sharing. Users do not mind what devices compose the system, they just need a specific functionality. System architects deal with selecting

1We extend the definition of COTS to include hardware devices
the most suitable devices (like lighting bulbs or a smart board in our case of study) for providing that functionality.

3 MDA for Pervasive Systems

Figure 1: MDA building blocks and our proposed techniques for pervasive computing

The MDA proposal defines the building blocks for constructing model driven methods, but it does not specify concrete techniques. In order to make MDA more useful, a MDA method should provide techniques for each MDA building block. Applying the MDA approach (see Figure 1) we propose the following techniques:

1. A precise language for building Platform Independent Models (PIMs). System developers use this language for precisely describing the system with high-level constructs. We propose to use the Pervasive Modeling Language (Perv-ML), described in section 4.

2. One or many modeling languages for building Platform Specific Models (PSMs). The constructs of these languages must be direct representations of constructs of the technology they model. This means that, for instance, a language for modeling an object oriented language must have elements like class, attribute, reference, etc. Currently, we are working in the development of a language for modeling an OSGi system. OSGi is a Java middleware initially created for hosting software of residential gateways.

3. PIM to PSM transformations. These transformations define how a PIM can be converted to a PSM. Currently, model transformations is a hot research topic. We apply graph grammars for defining the transformations from Perv-ML to OSGi.

4. PSM to source code transformations. Finally, the code generation from the PSMs is the last step of the development method. Due to the fact that PSMs are expressed using technological terms, transformation to source code is immediate. We are applying templates to the elements of models in order to obtain the source code.
4 PERVasive Modeling Language (Perv-ML): the PIM Language

Pervasive Modeling Language (Perv-ML) is a language designed with the aim of providing the system analyst with a set of constructs that allow to precisely describe the pervasive system. Perv-ML follows the approach described in section 2, promoting the separation of roles where developers can be categorized as analysts and architects. Systems analysts capture system requirements taking as an input the users needs and they describe the system at a high level of abstraction using three diagrams (models) that constitute what we call the Analyst View. On the other hand, system architects specify what COTS (devices and/or software systems) realize system services building other three specification models that constitute what we call the Architect View. Fig. 2 shows the language organization. The dashed arrow of Fig. 2 defines the construction order of the conceptual models that our approach proposes.

Figure 2: The six models of Perv-ML

We define the graphical notation of PervML by using the extension capabilities that UML provides. Next, we describe the main concepts of the language.

4.1 Modeling the Analyst View

Analyst describes the system using the services metaphor. We define a service as a coherent group of functionality defined in terms of operations. A pervasive system is built from a set of functional elements that provide a specific set of services that the user of the pervasive system requires. Those functional elements are what we call service instances. For instance, if the meeting room described above has two binds and any user wants to control them independently, the pervasive system must provide two elements (instances) that provide the bind management service. Following this approach we propose a step previous to the building of the Pervasive System Conceptual Structure. In this first step, we introduce the Services Model where the analyst defines services and their relationships in an abstract way without taking into account
the concrete service instances that finally represent our target Pervasive System.

The description of a type of service involves defining its interface and its semantics. Analyst defines the service interface specifying the operation that the service provides. In order to define the semantics, analyst specifies pre- and post-conditions for operations, integrity constraints and the behavior of that type of services. Moreover, analyst can define relationships between different types of services for specifying specialization/generalization or aggregation of services. Perv-ML uses and extends UML Class Diagram for representing the description of the services, and the State Transition Diagram for modeling the behavior. Fig. 3 shows the Service Model of our meeting room.

Analyst defines the pervasive system functional structure in the **Structural Model**. This model specifies the service instances of the system which are represented by a component. Perv-ML provides components as abstractions of the low-level elements that realize the services. Every system component provides one of the services described in the Services Model. When a component provides a type of service that aggregates other services, analyst must define in the Structural Model dependence relationships (depicted by dashed arrows) between such a component and other components that provide the aggregated services. In Fig. 4, we can see that the LightingManagement component has dependence relationships with the MainLighting and the Presence components due to the aggregation relationship defined in the Services Model. Perv-ML represents the Structural Model as a UML Component Diagram.

As we have said in section 2, system services must cooperate in order to satisfy all the system requirements. Analyst describes services cooperation in the **Interaction Model**. An interaction is a communication between services for providing a specific functionality, so analyst must describe as many interactions as joint functionality the system provides. Every interaction is described by an adapted UML Sequence Diagram, therefore the Interaction Model is composed by several sequence diagrams. For describing an interaction, analyst identifies the components of the Structural Model that participate in the interaction, defines the sequence of messages that the compo-

---

Figure 3: Meetings room Services Model

---

*Figure 3: Meetings room Services Model*
Figure 4: Components that provide the services of our case of study system.

Figure 5: An interaction that lowers blinds and sets lighting to 20% of its maximum intensity.

ments must interchange and specifies the condition (defined over property values) that triggers the interaction. The trigger describes the system state when the interaction must start.

Fig. 5 shows an interaction for suiting lighting when the blackboard service is being used. It lowers both blinds and it sets the lighting service at a 20% of its maximum power. This interaction takes place when somebody starts using the blackboard.

4.2 Modeling the Architect View

We need to build a detailed specification of the lower level artifacts that realize system services in order to have a complete and operative pervasive system description. Component interfaces in conventional software systems are implemented by classes (in object oriented approaches) or functions (in structured approaches). In pervasive systems, services are provided by artifacts that interact with the system environment, so an expert (the system architect) on those artifacts should specify component internals.

We use the term *Binding Provider* for referring artifacts that the pervasive system manages to interact with its physical or logical environment. A *device, a sensor, an*
Actuator or an external software system can be binding providers. A pervasive system can hold many similar binding providers, for instance, many lamps, many TVs, etc. Architect describes every binding provider type that is introduced to realize system services in the Binding Providers Model. A type of binding provider represents a set of devices or software systems that provide a similar functionality without detailing manufacturer specific information. Architect describes common functionality that a DVD player or a mail delivery system should provide. The description of a binding provider type is analogous to the description of a type of service in the Services Model. The Binding Provider Model is depicted using a stereotyped UML Class Diagram. Fig. 6 shows some binding providers of our meeting room. The usage of Lamp and FluorescentPanel actuators is different although both can be used for lighting a room.

The System architect uses the Component Structure Specification to specify the bindings providers that realize a component of the Structural Model. So, every system component in the Structural Model must have a Component Structure Specification where the architect includes bindings provider instances. For instance, a component that provides a lighting management service can be realized by three lamps and a fluorescent panel. In such a case, the Binding Providers Model must contain the lamp description. See Fig. 7 of the Structure Specification for the MainLighting component included in our meeting room Structural Model (see Fig. 4).

Finally, architect must specify how every component operation is realized. In the Component Functional Specification architect defines the sequence of actions that the component realize when an operation is invoked. These actions call operations or property values from:

- the binding providers that build the component,
• the service it provides,
• the components that it depends, in the case of having defined dependence relationships.

Architect specifies actions using the UML Action Semantic Language (ASL). ASL does not have an official concrete syntax, but many proposed syntaxes are available like the provided by Kennedy Carter [13].

Using the Perv-ML approach the system is completely described in a technology and manufacturer independent way. When a new technology emerges, system description does not need to be modified. Moreover, if we want to use a device of a new manufacturer we only have to develop a driver that adapts its interface to the generic interface used in the Binding Providers Model. Even if the system architect decides to change a component specification, analyst view remains unmodified. We have isolated changes by means of stratification through abstraction levels.

5 The Other MDA Building Blocks

As depicted in Fig. 1, a MDA method should define techniques to be used in order to build Platform Specific Model and to define model transformations between PIMs an PSMs. In this section we outline our approach.

5.1 OSGi Metamodel: the PSM Language

The Open Service Gateway Initiative (OSGi) [6] is an association of companies, that includes Sun Microsystems, IBM, Oracle and Nokia, created with the aim of developing an open standard for service gateways. A service gateway is the platform where resides the software for providing home services. It manages home devices and it communicates with external networks. The standard defines Java APIs for libraries that the OSGi platform provides. These libraries build a framework that offers high level abstraction constructs in order to build OSGi-based software. Therefore, it is a very suitable platform for developing pervasive systems.

In order to integrate OSGi in our development method, we have to be able to create models that are built using OSGi constructs. We have developed an OSGi metamodel for defining these constructs and their relationship. The models are represented using an UML profile. Fig. 8 shows a simple OSGi PSM that is depicted with a piece of the code that it generates. The mappings between the elements for the PSM and the OSGI code are one-to-one.

5.2 Graph Transformations. Defining the Model Transformation Engine

Define transformations between PIM and PSM involve jumping a wide gap between abstraction levels. Currently standards for transformation definition do not exist [2].
OMG published a Request For Proposal [7] in order to achieve a language for defining transformation between metamodels built with its Meta Object Facility (MOF).

In the meantime, we are using graph transformations and graph grammars [3] as the model transformation engine. There exist many works [1, 4, 11] that propose graph grammars as a suitable technique for model transformation.

From a mathematical point of view, a model can be seen as a graph where model elements are labeled nodes and the relationships between model elements are edges. In this way we can apply all the existing knowledge for defining graph transformations in order to achieve model transformations in the MDA context.

Fig. 9 shows two rules for model transformation from Perv-ML models to OSGi-based models. Every rule is composed by a Left Hand Side (LHS), that defines a pattern to be matched in the source graph, and a Right Hand Side that defines the replacement for the matched subgraph. For instance, first rule says that when a Perv-ML Component element is found it must be transformed into a Bundle element and references to a ServiceActivator and Manifest elements have to be created and linked to the Bundle. Following this approach, transformation from Perv-ML models to OSGi-based models is defined following a set of rules like those defined in this section.

```
import org.osgi.framework.*;

public class LightingManagement_Activator implements BundleActivator {
    private ServiceRegistration servReg = null;
    private ServiceReference lighting, presence;

    public void start(BundleContext bc) {
        lighting = bc.getServiceReferences("xxx.yyy.zzz.GradualLighting", "(PervMLid=MainLighting)\[0");
        presence = bc.getServiceReferences("xxx.yyy.zzz.PresenceDetection", "(PervMLid=Presence)\[0");
        Activation PC(lighting, presence);
        Properties props = new Properties();
        props.put("description", "Manages Room Lighting");
        props.put("HAMONid", "LightingManagement")
        servReg = bc.registerService("xxx.yyy.zzz", PC, props);
    }
}
```

Figure 8: A simple OSGi PSM and its related source code
6 Conclusions

In this paper we have introduced a method for pervasive systems development. This method has been developed in the context of the MDA proposal in order to provide a well defined path from high level pervasive system descriptions (using Perv-ML) to the source code of a concrete technology (OSGI).

Our current research focuses on developing tools supporting this method. These tools will guide system developers to systematically develop pervasive systems. They allow the developers to build conceptual models (PIMs) that constitute the source of an automatic code generator which applies model transformation rules. At the same time we are developing case of studies that help us to tune the method. In the future, we are planning to integrate this method in the SOSY Modeler CASE Tool (www.sosyinc.com / www.care-t.com), the industrial tool that supports OOMethod. OOMethod is a model driven method for developing information systems developed in our research group.

References


Model-Driven Methodologies for Pervasive Information Systems Development

José Eduardo Fernandes  
Dep. Informática e Comunicações  
Instituto Politécnico de Bragança  
Bragança, Portugal  
jef@ipb.pt

Ricardo J. Machado  
Dep. Sistemas de Informação  
Universidade do Minho  
Guimarães, Portugal  
rmac@dsi.uminho.pt

João Álvaro Carvalho  
Dep. Sistemas de Informação  
Universidade do Minho  
Guimarães , Portugal  
jac@dsi.uminho.pt

Abstract

This paper intends to introduce the concept of pervasive information systems (PIS) and the issues that arise from the software development for pervasive information systems. The model driven approach is generally described and its benefits to the software design are identified. Finally, some future directions for the usage of model driven methodologies within the development of PIS are highlighted, presenting some specific problems that nowadays that kind of methodologies have not yet been able to overcome.

Keywords: MDD, pervasive, information systems, software development

1 Introduction

Over the last years we have been assisting to a continuous growing trend for research and investment efforts in pervasive computing [19]. Benefiting from the growing of miniaturization and power of computing devices merged with improved wireless communication capabilities, information technology landscape is becoming more integrated with our real life landscape, allowing us to, probably in the present, and certainly in a near future, achieve some of the ambitions of the past regarding the integration of computing with reality, and the benefits derived from it.

Albeit the still common traditional use of computers as "distinct machines", they are being increasingly integrated (embedded) in other (non computing) devices, and consequently, not being seen or used as computing devices [6] [26]. Instead, the first look and thought goes to the larger (non computing) device as being a "smart" device.
Nevertheless, the embedded computing device is present, monitoring and processing information, and giving the output that it was conceived for.

The dissemination (embedding) of computing devices, with its particular computing and communications capabilities, into everyday physical objects (non computing devices), allows a real time access to information by those embedded computing devices. New advances in information technology (smaller computing devices, wireless networked communication systems, interface devices and others) stimulated imagination on ways to deploy this technology in service to people and organizations. Location and status of people and goods can be tracked in real time, and the appearance of "smart" goods, which beyond monitoring themselves, can also independently communicate with other computing devices or systems, allows for monitoring and interaction that were not possible in the past, therefore allowing the conception of new systems [19].

Pervasive computing, initiated a little more than a decade ago based on the seminal work of Mark Weiser [40], envisions an environment in which computers will be embedded in our natural lives, assisting us in an ubiquitous manner on our everyday tasks. Pursuing this vision, pervasive computing system engineers and researchers, started to design devices and systems that technologically proved the viability of this vision.

2 Pervasive information systems

Proved and assessed the value of pervasive computing, pervasive (information) systems started being devised for use in organizations. Beside social concerns [34] regarding the particular characteristics of pervasive computing, and economic implications of its deployment [19], new systems are being though, aimed at improving, not just people’s quality of life, but also the way business is done or even enabling new and innovative ways of carrying out business.

Museums [9], agriculture [8], restaurants [31], and health care [36] are examples of business domains that have been addressed by applications based on this kind of information technology. For example, in elder care [32] systems have been deployed, where pervasive computing improved the quality and the efficiency of care delivery to the elderly, assisting the staff in identifying people needing immediate attention, and to monitor trends, resulting in enhanced environment. Among the most relevant projects worldwide, the following can be emphasised: Active Badge [37], Active Campus [11], Aware Home [15], eClass/Classroom2000 [1], Labscape [4], iRoom [13], MediaCup [7], PARCTab [38], PlantCare [18], Smart-Its [12]. All these projects aimed to prove the viability and usefulness of the pervasive computing vision.

The benefits and applications of pervasive computing are far from having reached an end. Other business domains, such as insurance companies or government agencies can also take benefits of pervasive computing. What was initially confined in
developing technology to make pervasive computing out of a vision [20], surpassed the initially restricted frontiers to reach the development of applications for organizational domains, allowing the improvement of current business process or even to assist the development of new business models [19].

Along with all this, inevitably, we become aware of the presence flow and processing of information, not only by the individual computing devices, but also with a more deep significance, by the overall system that emerges from the interactions of all the computing devices, linking them together in a coherent fashion. We can then recognize the presence of a pervasive information system. Indeed, all these systems constitute, some form of information system. They gather, collect, process, store and produce information aimed at contributing to an organization or personal needs in order to achieve a set of well established objectives.

These pervasive information systems, composed of heterogeneous, mobile, or physically integrated (embedded) devices, capable of collecting, processing, and instantaneously communicating and interacting among them or others systems, opens the possibility of processing large quantity of information which, composing an information system, must be, in its essence, well designed, developed and deployed in order to, satisfying the requirements and exploring all the potential offered by the pervasive computing, maximize the revenue of these kind of systems. This is the way to efficiently satisfy the organizational or personal information needs in the pursuit of defined goals.

Research efforts so far have been mostly oriented, towards physical and virtual integration, interaction models, deployment, communication technologies and connectivity, and software architectures. It is important that these new pervasive systems also become subject of study from an information systems perspective.

Several pervasive computing characteristics, issues and challenges have been identified [2] [27] [20] [26]. Physical integration and spontaneous interoperability [16], quantity and heterogeneity of computing devices, services and applications the may be part at any moment of the system [10] are characteristics of pervasive systems that must be taken into account when designing PIS. They are important specially when aiming at easily reconfiguration of the application to cope with presence or absence of computing devices with distinct computing power or interface capabilities, availability of services, and to rapidly respond to business changing needs.

3 Model-driven software development

The high pace of technology innovation and changes, the growing complexity and the size of systems, the continued need of new or renewed systems with a short cycle time of development, puts pressure on the software development community which, on the need to cope with this reality and to bring higher quality and productivity, permanently searches for better development approaches and tools.

Model driven development (MDD) [28] [21], a recent advance in software de-
development, represents a new direction for software system development. It embodies a new manner of facing the development of software, essentially characterized by turning the models into the essence and focus of the development of a software system, independent of the possible platforms where they will be deployed. This change of focus, from code oriented to model oriented development, represents a significant evolution in software development, benefiting from an increase of abstraction, similar to what occurred, in the past, when through the introduction of compilers, there was a move in programming from assembly to 3rd generation languages [28].

The MDD approach allows for an abstraction of details of implementation platform technology and the use of concepts closer to the problem domain. Consequently, the models used to develop the system are easier to specify, understand and maintain, and are robust to changes of the technology adopted for system implementation [28].

Supporting this approach, widely accepted standards have emerged (and others will emerge), such as the Unified Modeling Language (UML) [25], and the Object Management Group (OMG)’s [23] Model Driven Architecture (MDA) [24] initiative that reinforces and promotes this new direction on software development.

Also fundamental to MDD, evolution on automation technologies enables to expect that the final system may be constructed through automatic code generation and other automatic facilities. This allows a reduction or elimination of the semantic gap existing in the traditional software development, consisting by the difficulty of mapping the concepts used to express models of the applications on the implementation technologies of the platforms [28].

MDD comes to improve and accelerate the software development, achieving a higher quality of the developed systems. It is expected that MDD may bring an increase on the portability, interoperability and reuse, together with an easier way to maintain and evolve a system.

The development of MDA tools and other related developments, along with the increasing deployment of MDD based projects, foster the movement toward model centric development [35]. Several research is being done in MDD field, from model transformation [30] [39], through separation of concerns [17] to executable models [22]. OMG contributes actively to the development of standards that helps MDD become a reality. In particular, UML (a key standard to MDA initiative) has been subject to revisions and evolutions, and its new version, 2.0 (to be ready during 2004), will represent a further impetus to MDD [29].

Model driven development is emerging as a major improvement in software development. Attending to the pervasive information systems’ characteristics, and face to the new advances in software development, it is legitimate to expect that the benefits inherent to model driven development can be brought to the development of pervasive information systems. Nowadays it is possible to identify several variants of the original MDA recommendations, where we can find some specific methodological characteristics in what concerns either the process model and the modelling approach [21] [5] [3] [14] [33] [22]. However, none of these methodological approaches
has explicitly dealt with the support of software design for pervasive information systems. This means that some work has to be done to adapt those recommendations to this application field.

4 Conclusions and future directions

Model-driven development of pervasive information systems must take into consideration aspects not only related with the global information system obtained from the cooperation of diverse, dispersed, integrated or mobile computing devices that in conjunction contribute to the achievement of the information system objectives, but also with the individual computing devices.

Model-driven development methodologies must be capable of producing pervasive information systems that are resilient to technological changes, rapidly absorbing new technology through the promotion of stability of models.

Models for pervasive information systems must be tailored to allow the dynamically construction and allocation of customized applications to heterogeneous computing devices, with different computational or interface capabilities.

Pervasive information systems can benefit from model-driven development approach to software developing, leveraging the capabilities of system composed by those computing devices. Several areas can be researched for this improvement, as for example:

- UML (or other languages) capabilities in improved model accuracy specification and its contribution towards model execution or model centric development
- Patterns and separation of concerns to enhance reuse of models and codifying best practices
- Model transformations and model-driven development methodologies concerning pervasive systems, and MDD supporting tools.

These concerns must be further researched in order to assure that model driven development actually brings all its potential benefits to pervasive information systems and their application domains.

These research aims at highlighting the attention that must be paid to information systems, instead of single computing devices, in what concerns the modeling and design of global solutions that apply pervasive computing principles. Pervasiveness is, indeed, an emerging property of information systems, and not of the individual computing devices; these are not pervasive, they are, typically, embedded in the environment or in other physical artifacts. The ubiquitous concept is associated with the services that pervasive information systems make available.
References


20


22

Model Driven Engineering: A Position Paper

Marcus Alanen, Johan Lilius, Ivan Porres and Dragos Truscan
Software Construction and Embedded Systems
Laboratories
Turku Centre for Computer Science
Lemminkäisenkatu 14, FIN-20520 Turku, Finland

Abstract

The Model Driven Approach (MDA) as supported by the Object Management Group (OMG) describes the structural requirements of an engineering discipline where models, instead of source code, comprise the primary artifact. Model Driven Engineering (MDE), as outlined by Stuart Kent, brings forth the dynamic aspects of engineering, where process adherence and rigorous commitment to analysis are equally important. As such, MDE has a broader scope than MDA. We discuss our position on MDE and its requirements on tools and technology, especially considering the dynamics of a model-based software development method. We demonstrate our approach with an example of the specification of an IPv6 router targeted to a customized processing architecture.

1 Introduction

Model Driven Engineering (MDE) tackles the elusive problem of system development by promoting the usage of models as the primary artifact to be constructed and maintained. The term was first proposed by Kent in [1] and is probably derived from the OMG’s Model Driven Architecture (MDA) initiative [2]. OMG’s MDA is based on the idea of platform independent models (PIM) and platform description models (PDM) that can be realized using a variety of middleware and programming languages into platform specific models (PSM). We understand MDE as a broader term that includes all models and modeling tasks needed to carry out a software project from beginning to end.

We consider that OMG’s vision of MDA, although valid, is just one of the possible scenarios in an MDE process. A PIM to PSM transformation may be a necessary task in an MDE context, but also PIM to PIM transformations, e.g. how a PIM representing some customer requirements can be transformed into another PIM that realizes those requirements. In our understanding, the main key concept behind MDE is that all artifacts generated during software development are represented using common modeling languages. As a consequence, software development can be
seen as the process of transforming a model into another until it can be executed outside its development environment. If we only study PIM to PSM transformations as described in the MDA approach instead of a more general framework, we may miss important issues and be unable to provide a general solution to a broader problem. This scenario occurs if we consider the OMG standards as an authoritative description of the way that we build software instead of as an authoritative description of the way that we represent our software.

The current OMG standards present a static and structural view of models. They define several standard modeling languages, e.g. what is a valid model in a given language using OCL constraints and how to store a model in a file using XMI [3]. However, they do not discuss how models are created or how models evolve. This may be explained by reviewing the origins of UML: it was developed as a method-independent notation to document software artifacts. UML can be used in combination with practically any software development method and, as a consequence, the OMG standards do not contain any reference or support for software development. We believe that the OMG standards should also consider the dynamic aspects of model development. This ranges from the basics of model evolution using algorithms for model transformation to more sophisticated reasoning about why a model transformation meets new requirements. We may consider that the Software Process Engineering Metamodel standard (SPEM) [4] addresses this issue. However, SPEM tells us how to document a process, while “planning and executing a project using a process described with SPEM is not in the scope [of the standard]”.

In this position paper we summarize our approach to defining Model Driven Engineering methods. We discuss the basic collection of concepts, methods and tools needed to support such methods. The ideas presented here have been implemented both in the SMW toolkit, and the newer Coral toolkit. We have verified the validity of these ideas by developing an instance of a MDE Method for developing protocol processing applications. The method has been applied to the specification and implementation of an IPv6 router both on a software platform, using the Java programming language, and on a hardware platform, using the TACO [5] protocol processing architecture.

2 Model-Driven Software Development Methods

We define a model-driven software development method as a software construction method where all the relevant information in the project is stored in some kind of abstract model. Model development is then carried out as a sequence of model transformations.

Model driven engineering is the result of the recent development on computer languages, awareness of the need of software development methodologies and the constant need to tackle larger and more complex development projects. These forces are not new. Indeed, we could use the same naming pattern to create terms such as
punched card driven development, to describe the development methods used when compiler time was a luxury, or source code driven development, to describe the methods used in Extreme Programming and many open source projects, where source code is the key artifact. However, we believe that MDE opens a window for new development methods and tools that are not available or are too expensive to implement in other approaches such as source code driven development. These tools and methods can take profit of the fact that the artifacts describing our software are stored in a standardized way and are, to a certain extend, independent of the implementation technology.

The description of a model driven engineering method should contain all the elements that are usually present in any software development method. It should describe which final deliverables and intermediate milestones should be produced, which language should be used to create the previous artifacts and which tasks we should perform, and in which sequence, so that we can effectively create the required artifacts. However, we consider that there exists two main differences in a model driven engineering method with respect to a traditional development method. First, all artifacts are represented using a well-defined modeling language. Secondly, and as a consequence, we can create tools that process and transform all the artifacts in our projects. Therefore, we will require that all tasks in a model driven engineering method should be performed with the assistance of specialized tools. In this context, a clear understanding of model dynamics is a prerequisite to define any MDE method.

We have identified a four-layer approach to model dynamics. Each layer depends on the functions provided by the layers beneath it. Every layer empowers the modeling environment with new dynamic aspects, which would not be possible by the lower layers alone.

- In our approach, layer 0 defines the basic model management possibilities. This consists of creation and deletion of model elements, modification of the various associations between elements, and the evaluation of model constraints. In essence, the power at this first layer is the power given by the Meta Object Facility (MOF) [6]. A good overview of basic model management, and the use of Python as a scripting language for e.g. model constraint evaluation is given in [7].

- Layer 1 acknowledges model evolution as a continuous temporal process. Here, versioning is the key element, whereby tools can support undo/redo facilities, displaying and calculating differences between models, merging of models from multiple collaborative sources and, finally, providing full revision control of the development process. Several of these issues are discussed in [8], [9] and [10].

- Layer 2 implements the desired behavior using interoperable tools with editors, and transformation rules. This requires a complete set of modeling standards for the various activities that developers can rely on.
1. *Queries* are applied on a model expressed in one language and returns a set of elements of the same model expressed in the same language.

2. *Model Transformations* are applied on a model expressed in a given language and either modifies the model in place, or creates a model, possibly expressed in a different language. The upcoming QVT standard from OMG addresses this problem. Model transformations conceive a plethora of new interesting questions and topics, such as transformation taxonomy, correctness-preserving transformations, consistency checking and/or verification.

3. *Code Generation*, although a form of transformation, is sufficiently different from a model-to-model transformation to merit its own classification. The goal is to produce suitable input to a second-stage compilation or analysis tool. The target language is not a metamodel.

The main difference between queries and transformations is that the queries are free of side-effects, meaning that when applied on a model they do not change the model in any way. Many examples of queries and model transformations are given in [11].

- Layer 3 includes *intent* in model development. It studies why changes are made in a model, and when the method in use *allows* us to make the change. Far too long has intent and process adherence been considered a second-class citizen in software engineering. We regard MDE as a possible savior, by enabling us to describe process methodologies, problem analysis, estimations, and testing frameworks together with an evolving platform description model, and with the consistent ways to describe models, metamodels, transformations and constraints in MDA. We are currently developing a MDE based flow which is targeted towards protocol processor design. The basic philosophy behind the flow has been discussed in [12]. The technical report [11] discusses the implementation of the flow.

## 3 Conclusions

In this position paper we have discussed the main ideas behind our research on Model Driven Engineering. More information on the current status of the project can be found on the MDE web-site [http://mde.abo.fi](http://mde.abo.fi), where you can also find both the SMW-tool (which is discontinued) and the new Coral toolkit.

### Acknowledgments

Dragos Truscan gratefully acknowledges the financial support for this work from the HPY and TES research foundations.
References


UML-based Security Measures of Software Products

Yanguo (Michael) Liu, Issa Traore
Department of Electrical and Computer Engineering
University of Victoria, PO Box 3055 STN CSC
Victoria BC V8W 3P6, Canada
Email: yliu, itraore@ece.uvic.ca

Abstract

Better security engineering practice requires incorporating security concerns in the whole software lifecycle from requirements capture to software product delivery and evolution. Software measurement represents an effective tool for improving software product quality. Metrics have so far been developed for a wide variety of software quality attributes such as reliability, performance, and complexity. Security remains an exception: little attention has so far been paid to software security metrics. We propose in this paper a new paradigm, named the User System Interaction Effect (USIE) model that can be used as a basis for software security measurement at the architectural level. A USIE model captures user interactions with the system, and as such it can be derived systematically from UML interaction diagrams. We illustrate our approach by presenting a confidentiality metrics that can be generated for a collection of UML sequence diagrams.

Keywords: Software security, security metrics, software quality, UML.

1 Introduction

System security is informally defined as the ability for the system to resist accidental or deliberate attacks [1]. Security is becoming more and more important in the development of software applications mainly due to the dramatic increase in the number of software-based attacks in recent years.

A way to improve software quality is to use metrics to guide the development process. Metrics have been successfully developed for a broad range of software quality attributes including reliability, performance, and maintainability [2, 3, 4]. The still immature field of software security remains as a notable exception [5]. Fenton identifies three classes of entities of interest in software engineering measurement, which are respectively products, processes and resources [4]. The main objective in this work is to develop a family of security metrics for software products. According
to the Information Technology Security Evaluation Criteria (ITSEC), the discipline of computer security is founded on three basic properties: confidentiality, integrity, and availability [7]. There are of course several other aspects such as authentication, access control and accountability, which can be related, in one way or other, to these basic properties. In this paper, we focus only on confidentiality and derive corresponding metrics by relating it to a high level UML behavioral model, namely sequence diagrams [6].

The rest of the paper is organized as follows. Section 2 discusses software security theory underlying our framework. Section 3 briefly overviews UML sequence diagram and describes how it can be used to develop a User-System Interaction Effect (USIE) Model – a new paradigm introduced in our work. Section 4 shows how confidentiality can be measured using USIE models. Section 5 illustrates a case study showing the effectiveness of our proposed metrics in software security evaluation. Section 6 concludes this paper and discusses future work.

2 Defining Confidentiality

Software security involves multiple dimensions, such as authentication, authorization, audit, confidentiality, integrity and so on. Some of these dimensions, for instance, authentication, authorization, and audit, can be specified as system functional requirements, thus they can be verified and tested as normal system functionalities. Others like confidentiality and integrity correspond to non-functional requirements in engineering secure software. Non-functional requirements are difficult to capture and analyze, and it is quite common that software systems are being developed without serious handling of non–functional security requirements.

Our understanding of software security foundation is consistent with the work of Jacob that defines software security in terms of user-system interactions [8]. We think that it is safe to say that security issues arise because of the occurrence of some user interactions with the system. An interaction may affect system security either in isolation or by interfering with other interactions. According to Jacob, “confidentiality is about limiting how much one user can infer about another user’s interaction with the system by making an interaction with the system themselves [8].” Hence, confidentiality is function of the interference occurring among interactions between different users and the system. So, possible measure of confidentiality may consist of evaluating this interference. We denote by $\text{Conf}(I_A \rightarrow I_B)$ the confidentiality of $I_A$ with respect to $I_B$, where $I_A$ represents an interaction of a user A with the system, and $I_B$ an interaction of a user B with the system. In other words, $\text{Conf}(I_A \rightarrow I_B)$ measures how much B can discover about $I_A$ via $I_B$. 

32
3 Behavioral Modeling

3.1 Overview of UML Sequence Diagram

Interactions are described in UML using the so-called interaction diagrams, which consist of sequence and collaboration diagrams. Sequence and collaboration diagrams are dual diagrams. A sequence diagram can be derived systematically from a collaboration diagram and vice-versa. So, we’ll focus our attention on only one kind of diagram in this paper, namely the sequence diagram.

A UML sequence diagram is used to describe interactions between objects in terms of sequences of messages they may exchange as the interaction unfolds over time to effect the desired operation or result [6]. More precisely, it contains a set of role entities and a set of partially ordered messages, each specifying a communication between two role entities. In order to illustrate the concepts introduced in this paper, we use as a running example a subset of the requirements of a medical information system for patient records keeping. Figure 1 shows a UML sequence diagram that demonstrates a “ReadRecord” interaction between a doctor and the software system for keeping record. We assume that the medical records are stored in their encrypted form as a second layer of defense. This is not unusual with highly sensitive documents (e.g. password files etc.). From Figure 1, any useful read scenario involves obtaining a key from the key server and decrypting the record. The record is then re-encrypted before storing.

![Sequence diagram for “ReadRecord” interaction](image)

3.2 User-System Interaction Effect Model

It appears from the security definitions given in Section 2, that the understanding of user interactions, and the data and role entities involved play an important role in
the analysis of security events. Regular UML sequence diagrams, however, provide only the communication information for an interaction while the responses of role entities are not expressed. Moreover, analyzing the interference between several interaction sequences may require merging them in one view, which is not supported by existing UML interaction diagrams. In order to facilitate analysis of software security events, we introduce a new paradigm named User-System Interaction Effect (USIE) model that provides an abstract view of security relevant information carried by standard UML interaction diagrams. The abstract nature of USIE models allows combining information collected from several UML interaction diagrams in a single view facilitating security analysis.

To enable automatic derivation of USIE models from sequence diagrams, we extend the latter (in a standard fashion) by defining stereotypes that describe communication effects associated to the interactions. Hence, a USIE model captures, in one hand a trace of the communication in a user-system interaction as defined by a sequence diagram, and on the other hand highlights the communication effects expressed by the particular communication stereotypes.

A USIE model consists of a graph in which there are two kinds of nodes named InteractionStart and RoleEntity. An InteractionStart node represents the starting point of an interaction, and is in principle named after the interaction name (i.e. the name of the UML sequence diagram). A USIE model involves a single InteractionStart node. In contrast, a USIE model can have multiple RoleEntity nodes. A RoleEntity node corresponds to a role entity contained in the sequence diagram; it is named after a corresponding role entity name. The nodes of a USIE model can be derived directly from a corresponding sequence diagram.

The edges in a USIE graph have four characteristics. First, all the edges are directed. A directed edge in a USIE model represents a single communication (sending an event or invoking a method) between two role entities; it is directed from its source role entity to its target role entity. Second, an edge may optionally have attributes associated with it. There are two kinds of attributes named ChangeState and ReturnInformation. An edge has ChangeState attribute if the communication modifies the state of its target role entity; it has ReturnInformation attribute if the communication returns information to its source role entity. An edge can have zero or more attributes. Third, the edges in a USIE model are ordered according to the communication order expressed in the UML sequence diagram. Fourth, each node has exactly one incoming edge. In practice, the edge attributes, ChangeState and ReturnInformation, are expressed in a UML sequence diagram using predefined communication stereotypes. Therefore, the edges information of a USIE model can be derived automatically from the communication expressions of the corresponding sequence diagram.

We represent a USIE model using a graphical notation whose features are described in Figure 2. Figure 3 shows a USIE model that is derived from the sequence diagram of Figure 1 with all communications stereotyped. The edges in Figure 3 are ordered by numbering them from 0 to 4.
Figure 2: Graphical notation for USIE elements

Figure 3: Example of USIE Model
4 Security Measures based on USIE Model

4.1 Basic Notation

We define some basic notations that will be used to define our security metrics. Let $U = (N, E, <)$ represents a USIE model, where $N$ is a set of nodes, $E$ is a set of edges, and $<$ is a partial order relation over $E$. Each node in the USIE model must be named; we denote a node of the model by $N_a$, where $a$ is a string representing the node name. Since edges in the USIE model are ordered, we denote an edge of the model by $E_i$, where $i$ is a non-negative number representing the order of the edge. Given nodes $E_i$ and $E_j$, $E_i < E_j$ if and only if $i < j$; we say that $E_i$ comes before $E_j$. We use several additional notations that are listed in Table 1 with their interpretations.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_a : T$</td>
<td>Node $N_a$ of type $T$ (e.g. RoleEntity, InteractionStart)</td>
</tr>
<tr>
<td>Source($E_i$)</td>
<td>Source node of edge $E_i$</td>
</tr>
<tr>
<td>Target($E_i$)</td>
<td>Target node of edge $E_i$</td>
</tr>
<tr>
<td>Attributes($E_i$)</td>
<td>Attribute set of edge $E_i$</td>
</tr>
<tr>
<td>Incoming($N_a$)</td>
<td>Incoming edge of Node $N_a$</td>
</tr>
<tr>
<td>$N_a \cdot U$</td>
<td>Node $N_a$ of $U$</td>
</tr>
<tr>
<td>$E_i \cdot U$</td>
<td>Edge $E_i$ of $U$</td>
</tr>
</tbody>
</table>

4.2 Confidentiality Measures

In this section, we present a method to evaluate quantitatively the confidentiality $Conf(I_A \rightarrow I_B)$ of an interaction $I_A$ with respect to another Interaction $I_B$.

Confidentiality is related to information sharing, information leakage decreases confidentiality [8]. We measure confidentiality by identifying the potential information leakages between interactions. According to Kemmerer finding a pair of “Read” and “Write” operations on a shared resource could identify an information leakage channel between two processes [9]. More specifically, whenever a process may read a resource that another process can write, the former process may deduce some information about the latter process, which decreases the confidentiality of the latter process. In the USIE paradigm, a “Read” operation corresponds to an edge with ReturnInformation attribute; a “Write” operation corresponds to an edge with ChangeState attribute. Under these settings, we define an Information Leakage Channel as follows:
Definition 1. An Information Leakage Channel from a USIE model $U_1 = (N_1, E_1, \prec)$ to a USIE model $U_2 = (N_2, E_2, \prec)$ via node $N_a$, denoted $ILC_{N_a}(U_1, U_2)$, exists if

(a) $(N_a : RoleEntity) \land (N_a \in N_1 \cap N_2)$

(b) $ChangeState \in Attributes(\text{Incoming}(N_a \bullet U_1))$

(c) $ReturnInformation \in Attributes(\text{Incoming}(N_a \bullet U_2))$

An information leakage channel exists between two USIE models $U_1 = (N_1, E_1, \prec)$ and $U_2 = (N_2, E_2, \prec)$, whenever they share a common RoleEntity node, which is targeted in one model by a ChangeState edge, and in the other model by a ReturnInformation edge.

Information leakage channels can be categorized according to their importance. An information leakage channel between interactions is significant if the information leaked by this channel can be sent to the InteractionStart node, otherwise the channel is secondary. We give the following definition:

Definition 2. Let $ILC_{N_a}(U_1, U_2)$ be an information leakage channel via node $N_a$, then the channel is significant, denoted by $SILC_{N_a}(U_1, U_2)$, if

$$\forall E_i \bullet U, E_i \leq \text{Incoming}(N_a \bullet U_2) \Rightarrow ReturnInformation \in Attributes(E_i)$$

Figure 4 shows an example of leakage channels from an interaction $I_A$ to an interaction $I_B$. Channels labeled “S” represent significant channels; channels labeled “S” represent secondary channels.

When we evaluate $Conf(I_A \rightarrow I_B)$, we are primarily concerned by the significant information leakage channels. Based on this consideration, we propose for $Conf(I_A \rightarrow I_B)$ the following definition:

$$Conf(I_A \rightarrow I_B) = \frac{1}{1 + NOSILC(I_A, I_B)}$$

Where “$NOSILC(I_A, I_B)$” represents the “Number Of Significant Information Leakage Channels” from $I_A$ to $I_B$. Based on this formula, the confidentiality of $I_A$ with respect to $I_B$ is equal to 1 only if no significant information leakage channels exist from $I_A$ to $I_B$. The confidentiality decreases as the number of significant channels increases. For instance, in Figure 4, $Conf(I_A \rightarrow I_B) = 0.5$.

5 Case Study on Race Condition

5.1 System Design

Race conditions are mostly known for robustness problems they create in concurrent programs [10]. Some race conditions are, however, sources of serious security bugs [11]. A security attack based on race conditions exploits a window of vulnerabil-
Figure 4: Information leakage channels from $I_A$ to $I_B$

ity between event executions in order to force the system to behave in unanticipated ways. As an example, let us consider the record keeping system introduced in Section 3. Assuming that the medical records are stored in their encrypted form, an intruder who is able to break into the system using, for instance, a password-cracking tool would also have to decrypt the record before being able to do any kind of useful job with it. Let’s assume that an intruder can access the (file or database) server storing the records but not the keys server. Hence, the intruder is able to access the encrypted records but cannot decrypt them. The intruder, however, can still run an attack tool concurrently with the record-keeping tool that repeatedly tries to access the records. Figure 5 shows a sequence diagram describing such interaction from the Intruder’s perspective. Figure 1 describes the interaction from the doctor’s perspective. If there is no concurrency control mechanism in place, two possible schedules among others that may occur are the following:

1. \{decrypt(), view(), read(), encrypt()\}
2. \{decrypt(), read(), encrypt(),view()\}

Schedule (2) is acceptable but not schedule (1). In schedule (1), the attacker takes advantage of a window of vulnerability to access the record in cleartext. An alternative design, which is more secure, may introduce a concurrency control mechanism that will prevent unauthorized access to the cleartext file. Figures 6 and 7 show the sequence diagrams describing the doctor and intruder’s perspectives under this scenario.
Figure 5: Concurrent Read Record Interaction by an Intruder

Figure 6: Intruder Read record under Concurrent Control
5.2 Security Measures

We apply in this section our confidentiality metrics to both the non-secure and secure designs, and show that it can capture the difference between them. For instance, based on the non-secure and secure designs presented in section 5.1, two sets of USIE models that are shown in Figures 8 and 9 can be derived.

In Figure 8, two significant information leakage channels exist for the “DoctorReadRecord” interaction with respect to the “IntruderViewRecord” interaction. In Figure 9, since the secure design forces secure concurrency checking, the intruder’ “View” action of the secure system cannot proceed further while a record is being accessed. Hence, no channels exist for the “DoctorReadRecord” interaction with respect to the “IntruderViewRecord” interaction.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Non-Secure System</th>
<th>Secure System</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Conf}(I_{\text{DoctorReadRecord}} \rightarrow I_{\text{IntruderViewRecord}})$</td>
<td>0.67</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2: Security Metrics

Based on section 4, we may derive confidentiality security metrics for both the non-secure and the secure system designs. The metrics are shown in Table 5.2. As we discussed in section 5.1, the secure system design improves system security by preventing unacceptable schedules such as $\langle \text{decrypt()}, \text{view()}, \text{read()}, \text{encrypt()} \rangle$ from happening. Our security metrics reflect the security improvement by showing that the legal user’s (e.g. a doctor) interaction with the system preserves confidentiality with
Figure 8: USIE models of the non-secure design

Figure 9: USIE models of the secure design
respect to an intruder’s interaction during concurrent executions.

6 Conclusions and Future Work

Security awareness is increasing in the software community. This is a positive evolution because software flaws such as buffer overflows and race conditions are the sources of the most popular security attacks. We have presented in this paper a new paradigm that can be used as a basis for software security measurement at the architectural level. The approach has been illustrated by generating confidentiality measures for UML sequence diagrams. Future work will consist of extending our metrics framework by defining measures for other software security attributes and by also investigating other UML diagrams. We also plan in the future to conduct empirical and theoretical validation of our security metrics.

References


Modelling Message Based Real-Time Systems with UML and Rate Monotonic Analysis

Ian Oliver and Juha Savolainen

Nokia Research, Itämerenkatu 11-13, Helsinki Finland
ian.oliver,juha.savolainen@nokia.com

Abstract

Rate Monotonic Analysis for schedulability is a well known and proven technique for determining the timing related behaviour of a reactive system under various load - usually worst case execution time - situations. Certain kinds of telecommunication system however require the model of the system to be driven by message type rather than the plain event type stimuli that RMA usually uses. We describe here how to describe such systems using the UML as the description or modelling language within an task based modelling context that facilitates rate monotonic analysis.

1 Introduction

Schedulability analysis is a useful technique for the investigation of whether a system can meet various hard deadlines and for investigating the load upon a given system under various circumstances.

With the move to more complex embedded, real-time systems we are faced with the need to move from simplistic round-robin, event-loop type scheduling and message handling to more complicated scheduling systems and communicating processes/tasks. This also implies that various resources of the underlying platform are requested in complex ways leading to the inevitable problems of deciding process/task priority, priority inversion and the impossibility of easily predicting whether a given system will perform as expected.

In this paper we present firstly an overview of the UML-RMA mapping from a profile of the UML for Schedulability, Performance and Time (UML-SPT) [8] for describing and mapping certain kinds of timing information from UML to a task based Rate Monotonic Analysis formalism. This mapping was developed for the schedulability analysis of certain kinds of DSP software.

Using this as a base notation and mapping (and thus semantics) we then extend this to take into consideration the description and analysis of models where the primary driver is the data content of the message for schedulabilty analysis. We then
extend this to cater for the case where a pool of tasks are available for processing said messages.

2 UML and RMA

In [7] a profile for the UML-SPT was described that enables the modeller to annotate certain UML diagrams - class, sequence and state - with timing information within the context of certain kinds of reactive systems [9, 10]. This information can then be mapped to a task based formalism for subsequent analysis with any suitable technique, in particular Rate Monotonic Analysis (RMA). We also assume a refinement ordering between the information contained within the diagrams such that class diagrams are refined by sequence, sequence by state.

Here we concentrate on the mapping between the UML model and the task based model for RMA. This section does not provide a complete overview of the profile which can be found in the aforementioned [7]. The profile in summary can be seen in tables 2 and 2.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPactive</td>
<td>Class</td>
<td>Class with own thread of control</td>
</tr>
<tr>
<td>COP_PDC</td>
<td>Class</td>
<td>Class encapsulating data or functionality</td>
</tr>
<tr>
<td>COP_CPU</td>
<td>Device</td>
<td>Denotes a processing device, eg: CPU</td>
</tr>
<tr>
<td>COP_device</td>
<td>Device</td>
<td>Denotes a non-processing device, eg: Memory</td>
</tr>
<tr>
<td>COP_thread</td>
<td>Class</td>
<td>Denotes a thread or similar mechanism</td>
</tr>
</tbody>
</table>

Table 1: “Copenhagen” Profile Stereotypes

In addition we define a refinement relationship between the diagrams such that a class diagram’s timing information is refined by a sequence diagram’s, this then by a state diagram’s information. Finally if necessary any information attached to a method (on a class) refines everything else. Timing information is attached to classes to provide a constraint across all instances and contained features of that class. The message processing semantics is based upon FIFO queues and run-to-completion.

3 Mapping UML to RMA

While no specific notation for noting RMA models exists, we have chosen to default on the notation used in the TriPac RapidRMA tool. The RMA model can be considered as a directed, non-cyclic graph. We supplement the RMA notation by writing the letters a, s, r or f (asynchronous, synchronous, synchronous return and function call respectively) alongside the arrow to denote the type of message being sent between tasks.
There are four basic cases for the mapping of the UML sequence diagram to the RMA model. The first case we consider is that of a simple execution block where this maps to a single RMA task. In the second case (fig.1) is a function call message between two objects. It is normally the case here that all components of the execution are bundled together to form a single RMA task.

In the third case (fig.2) are asynchronous messages. It may be the case here that the two RMA tasks related by the function call message can be joined as in the case in figure 1. However this is really decided by the underlying execution mechanism discussed later.

The fourth case in figure 3 we show a synchronous message between two objects. Note that the execution block a2 can not be executed until execution block b has at least started to execute [4].

In figure 4 we can see the class and state diagrams responsible for the structure and behaviour of the system we are modelling - primarily however we are interested in the state diagram’s timings and especially the transition timing of 8ms,16ms. This

<table>
<thead>
<tr>
<th>Tag</th>
<th>Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPcontextswitchtime</td>
<td>COP_CPU</td>
<td>Processor context switch time</td>
</tr>
<tr>
<td>COPcpucycletime</td>
<td>COP_CPU</td>
<td>Time of 1 cpu cycle</td>
</tr>
<tr>
<td>COPrate</td>
<td>COP_CPU, COP_PDC</td>
<td>Rate of device</td>
</tr>
<tr>
<td>COPaqttime</td>
<td>COP_PDC</td>
<td>Aquisition time</td>
</tr>
<tr>
<td>COPdeaqtime</td>
<td>COP_PDC</td>
<td>Deaquisition time</td>
</tr>
<tr>
<td>COPonDevice</td>
<td>COP_PDC</td>
<td>Denotes on which device the PDC is implemented</td>
</tr>
<tr>
<td>COPonCPU</td>
<td>COPactive</td>
<td>Denotes on which CPU the Class is implemented</td>
</tr>
<tr>
<td>COPonThread</td>
<td>COPactive</td>
<td>Denotes on which thread the Class is implemented</td>
</tr>
<tr>
<td>COPhasThreads</td>
<td>COP_CPU</td>
<td>Denotes which threads run on the CPU</td>
</tr>
<tr>
<td>COPstateentry</td>
<td>States</td>
<td>State entry time</td>
</tr>
<tr>
<td>COPstateexit</td>
<td>States</td>
<td>State exit time</td>
</tr>
<tr>
<td>SAExecutionTime</td>
<td>any period of execution</td>
<td>Execution Time in cycles or time</td>
</tr>
<tr>
<td>SAOccurrencePattern</td>
<td>Messages</td>
<td>Rate at which message occurs</td>
</tr>
<tr>
<td>SADeadline</td>
<td>Messages</td>
<td>Deadline for complete processing of message</td>
</tr>
</tbody>
</table>

Table 2: “Copenhagen” Profile Tagged Values
timing figure must be a refinement of the sequence diagram timing figure for that message between those two particular states.

In figure 5 we can see the sequence diagram detailing the execution involved upon receipt of message $t$ between states $s_1$ and $s_2$. Given this sequence diagram we obtain the RMA task graph as shown in figure 6. If we now take into consideration the state diagram from figure 4 we obtain the RMA task graphs shown in figure 7. Note that we can treat the entry and exit times of the states and being analogous to function calls. Note also that the total timing of the state diagram transition plus state entry and exist times must be less than or equal to the total timing given on the sequence diagram.

Note in figures 6 and 7 we may see the RMA task graph collapse with the unification of those tasks joined via function call messages as per figure 1.

We have discussed how the task specifications map to the RMA world but not explicitly how the hardware or deployment mapping is made. We devised no notation for hardware in RMA as the information contained in the model was normally incorporated into the RMA task timing information.

For each ‘device’ and thread we must create corresponding logical semaphores in order to preserve the fact that the logical cpus/devices can only support one executing task/access at any one time. In most RMA tools this is assumed for CPUs. It is possible to support non-binary-semaphores so that multiple access to devices is possible - this situation is however rare in our experience and normally we do not support this mode of operation for logical semaphores.
In previous sections, we have assumed, along with the general RMA theory [1], that each task can be activated by only one type of an event. This is typical to many control systems. A task is allocated for each actuator or device that is capable to create events. This structuring of tasks allows easy determination of task priorities, since varying deadlines among the events can be assigned to the priority of the corresponding task. This is also a basis of some real-time engineering methods [2, 3].

The RMA approach seems to give good results for real time control systems and should suit well any system that has many independent threads of control or has been decomposed along the lines of interaction. However, many systems in the telecommunications domain do not fit into those categories. Many network elements share the same basic behavior. They basically encrypt and decrypt, package and unwrap, multiplex and separate data into common data channels.

In practise, this means that many different message types travel through the same channel. At the same time, the actual processing and allocation of processing to running tasks heavily depends on the type of the message being processed at a given time. One task is typically responsible for delegating the processing to other tasks based on the message type. This effectively creates paths of execution through the system, which are partly independent, and for other parts share some of the same tasks. For those tasks that are only used by one type of message the analysis is straightfor-
ward and presented in the previous sections. In this section, we focus on modelling the tasks that are shared by the multiple message types. Telecommunication system deadlines are often determined as required end-to-end response times.

### 4.1 Event Type Specific Modelling

Using the modelling patterns leaned from the previous sections we can now transform the MSC described in the figure 8. To the task model shown in the figure 8.

Here we have four tasks that are interdependent. They perform some processing and then forward the execution to the next task. Finally, the results are passed back via the same tasks that participated the processing. The resulting task structure is shown in the figure 9.

In this case, we have assumed that each of the tasks do not perform any processing after they have send the asynchronous event to the next task. If some processing would take place we would add new tasks to represent the remaining execution.

Now consider having two messages of different type. Both of these messages will pass through tasks A, B, and C. But depending on the message type the messages specific processing will take place either in task D or C. The figure 10 shows the MSCs of both of the cases.

The event 1 results an execution of tasks A, B, C, and D while the event 2 uses the same first three task but is finally passes to task E. Along with the previous example
we now have two different task structures where the tasks A, B, and C have been
duplicated to represent the two different event types. Tasks D and E appear only once
in the event specific task structures. The characteristics of the duplicate tasks, like
A11 and A12 that correspond to event 1 and event 2 respectively, remain the same.
The estimated execution time of the task A is copied to the tasks A11 and A12.

The final task structure that covers both message types has the total of eight tasks.
In this model the also the characteristics of the events remain the same. There is no
change to the deadline or the periods of the events.

4.2 Process Pools

Process pools are a typical way to have dynamic allocation of processing resources.
The pools are a well-known way to manage dynamically changing need for resources
[5, 6]. They greatly reduce the initialisation overhead compared to dynamical adding
and removing resources without a pool of statically initiated processes.

In order to handle a message, the process C selects a free process from the pool
of 20 processes. In the simplest form of the process pool design the next free process
is always chosen. However, the process pool pattern allows also having a segmented
pool that has varying process priorities based on the importance of the message. How-
ever, in this case we consider that only one type of message requires resources from
the process pool. The resulting task structure is shown in the figure 12.

In figure 13 we have multiplied the tasks A, B, and C by the number of processes
in the process pool. This creates as many independent task structures as there are
processes in the process pool. Again the characteristics of the duplicated tasks are identical to the processes in the MSC. That is, the task A1x has the same properties and the task A. The tasks D1-D20 have the characteristics of the individual processes in the process pool. However, in this case the event properties change. Input period of the events is 1/20 of the actual frequency, since the tasks structures are multiplied. In general, this means that the event frequency of individual task structures must be divided by the number of the processes in the process pool.

5 Example

In this section we present a short example based upon a situation occurring in some real telecoms device. Timing values and task names have been simplified for this example.

There exist two types of message, the first occurring every 1137ms with deadline 100ms and task interaction as shown in figure 14. The second message causes a task interaction as shown in figure 15 with period 500ms, deadline again of 100ms and utilises a process pool of 5 processes. All tasks exist executing on a single processor within a single thread. We do not consider resource contention in this example for simplicity.

This can be translated to a tasking model as per the UML-RMA mapping discussed earlier and also taking into consideration the processor pool allocation. We have 5 processor pools available, this gives us for the second message type 5 copies of the task structure and a multiplied period of 2500ms per instance of the message.
Figure 11: Task Structure (2)

Figure 12: Four Task Model with Process Pool

In total 42 tasks exist in the tasking model as described below with 6 resulting task structures (1..6):

\[
(A_1 \to B_1 \to C_1 \to F_1 + F_2 \to C_2 \to B_2 \to A_1) || (1)
\]

\[
(A_{11} \to B_{11} \to C_{11} \to (D_{1} + E_{1} + D_{2})_{1} \to C_{11} \to B_{11} \to A_{11}) || (2)
\]

\[
(A_{12} \to B_{12} \to C_{12} \to (D_{1} + E_{1} + D_{2})_{2} \to C_{12} \to B_{12} \to A_{12}) || (3)
\]

\[
(A_{13} \to B_{13} \to C_{13} \to (D_{1} + E_{1} + D_{2})_{3} \to C_{13} \to B_{13} \to A_{13}) || (4)
\]

\[
(A_{14} \to B_{14} \to C_{14} \to (D_{1} + E_{1} + D_{2})_{4} \to C_{14} \to B_{14} \to A_{14}) || (5)
\]

\[
(A_{15} \to B_{15} \to C_{15} \to (D_{1} + E_{1} + D_{2})_{5} \to C_{15} \to B_{15} \to A_{15}) || (6)
\]

Each individual task $A_x, B_x, C_x, E_x, G_x$ has execution time of 1ms. Task $F$ has execution time of 4ms plus a synchronous or function call to task $G_x$ totalling 5ms. Similarly Task $D$ has execution time of 10ms plus a synchronous or function call to task $G_x$ totalling 11ms.

Analysis using the RMA tool RapidRMA\(^1\) results in a total CPU utilisation of 90.29% with schedulability being attained at this loading. This represents a typical

\(^1\)www.tripac.com
high load or even worst case scenario under these message conditions. In figures 16 and 17 we can see screenshots of the RMA tool with this analysis.

6 Conclusions

We have described in this paper an overview of the mapping from the Unified Modelling Language to a task based formalism suitable for analysis with Rate Monotonic Analysis techniques.

Using this as a basis we have presented a way of mapping systems that are driven by message type rather than plain event stimuli. This is particularly important in certain kinds of telecoms system.

This is then enhanced by allowing the specification of processor pools to take into consideration the situation where a number of individually running copies of a task await the incoming messages. We have then described briefly how this may be dealt with by RMA. Finally we demonstrated our approach using a simplified example.

Our extensions were created out of a very practical need to analyse the schedulability of a real network element. We strongly believe that our results are useful for other practitioners operating with systems driven mainly by messages.
References


Figure 16: RMA Task Model

Figure 17: RMA Analysis
A Partition Methodology to Develop Data Flow Dominated Embedded Systems

António J. Esteves and Alberto J. Proença
Department of Informatics, University of Minho
Braga, Portugal
{esteves,aproenca}@di.uminho.pt

Abstract

This paper proposes an automatic partition methodology oriented to develop data flow dominated embedded systems. The target architecture is CPU-based with reconfigurable devices on attached board(s), which closely matches the PSM meta-model applied to system modelling. A PSM flow graph was developed to represent the system during the partitioning process. The partitioning task applies known optimization algorithms - tabu search and cluster growth algorithms - which were enriched with new elements to reduce computation time and to achieve higher quality partition solutions. These include the closeness function that guides cluster growth algorithm, which dynamically adapts to the type of object and partition under analysis. The methodology was applied to two case studies, and some evaluation results are presented.

Keywords: partitioning, hardware/software co-design, PSM meta-model, tabu search, cluster growth

1 Introduction

This paper describes an automatic partition methodology oriented to develop data flow dominated, medium complexity and real time embedded systems, where a processing element coupled to FPGA/CPLD board(s) [1] form a reconfigurable architecture [2]. Since this architecture includes hardware and software components, the present work applies the hardware/software codesign paradigm.

Partitioning is an NP-complete optimization problem that assigns system objects to the target architecture components and defines its startup time (scheduling), to achieve the designer objectives quantified by a cost function [3]. The partition process converts an unified and uncommitted system representation to a multi-part representation committed to the target architecture components. The present approach performs a functional, inter-component and automatic partition.

The partition task is part of a development methodology that covers all phases of systems development [4]. It is based on an operational approach [5], it runs at a high
abstraction level and it takes advantage of the object oriented modelling paradigm to reduce complexity and design time. Common to object oriented approaches, it uses multi-view modelling to describe the objects, the dynamic and the functional perspectives of systems. Following an operational approach, an executable specification is developed, which runs through a set of refinements and transformations to achieve a system implementation. When compared to the methodology followed by the MOOSE approach [6], the proposed methodology has some advantages: (i) the state transition diagrams (STD) are replaced by PSM\textsuperscript{1} models [7], which allow adequate handling of the system objects concurrency, (ii) implementations follow an iterative approach, replacing the traditional cascade design flow and (iii) the partition is automatically performed, without requiring additional expertise from a codesign professional.

To evaluate this partition methodology, a prototype tool was implemented, parTi-Tool, and its capabilities were compared to other approaches, following the structure introduced in [8]. Here, two sets of features are grouped for comparison purposes: the modelling support and the implementation support. The first identifies 3 axis: the application domain (control, data or data+control), the type of validation (simulation or co-verification) and the modelling style (homogeneous or heterogeneous). Figure 1 shows where parTiTool fits in the graph and how it relates to other approaches. Most approaches adopt homogeneous modelling style, where the only allowed validation method is simulation. Systems are described with a software oriented language (C, a C variant, Occam or C++) or an hardware oriented language (VHDl, Verilog or HardwareC). The proposed approach is part of a development methodology that uses heterogeneous modelling. It can be applied to data and control systems, but it is oriented to data flow dominated systems. In the present stage of the evolution, it does not allow co-verification.

To compare the support available to implement the systems with multiple components, figure 2 also uses 3 axis: the support to synthesize the interface between components, the supported target architecture and the automation degree of the partition process. A reasonable number of approaches (Chinook [9], Cosmos [10], CoWare [11] or Polis [12]) does not execute partition automatically. None of the approaches completely supports the automatic partition and the synthesis of interfaces. In the proposed approach the partition process is automatic and the information required to synthesize the interface between components can be extracted from the detailed model used to estimate communication metrics. Current parTiTool prototype implementation does not support yet more than one microprocessor, due to the target evaluation architecture. However, a R&D track is being prepared to merge this project with current adaptive load and data scheduling in parallel and distributed systems [13].

The paper is organized in 4 sections. Section 2 describes the proposed partition methodology, namely the formal description of the partition process, the approach

\textsuperscript{1}Program-State Machine.
followed to model the system and its internal representation, the construction and improvement of partition solutions with cluster growth and tabu search algorithms and the metrics estimation required by the evaluation functions. Section 3 presents the prototype system used to validate the partition methodology validation – target architecture and applied tool – and summarizes the case studies and the obtained results. Section 4 closes with conclusion remarks and directions for future work.

2 Partition methodology

The presentation of the partition methodology starts with the formal description of the partition process. Given an unified representation for the system (see below, in system modelling), the partition process generates a description for each component of the target architecture to be used on the implementation of the system. To reach this goal, the set of objects on system description must be divided into a series of disjunct sub-sets that will be assigned to the different components of the target architecture. The task that divides the set of objects on sub-sets is guided by the target architecture.
constraints and the design requirements. In the present work, the objects represent program-states or variables from the system PSM model. A formal definition of the partition process follows below.

Given the set of objects \( O = \{o_1, o_2, ..., o_n\} \) that models the system functionality, the set of constraints \( Cons = \{c_1, c_2, ..., c_m\} \) and the set of requirements \( Req = \{r_1, r_2, ..., r_p\} \) that define the feasibility and the quality of the partition alternatives to be generated, the partition process generates several sub-sets (or partitions) \( H_1, ..., H_{nh}, S_1, ..., S_{ns}, \) where \( H_i \subseteq O, S_i \subseteq O, \{H_i\}_{i=1..nh} \cup \{S_j\}_{j=1..ns} = O, \) \( H_i \cap S_j = \emptyset, H_i \cap H_k = \emptyset \) (with \( k = 1..nh \) and \( k \neq i \)) and \( S_j \cap S_l = \emptyset \) (with \( l = 1..ns \) and \( l \neq j \)).

The selection of a partition solution, among all that were analyzed by the partition algorithm, implies a cost function \( F_{\text{cost}} \). This function uses the sub-sets of objects assigned to hardware \( H = \{H_1, ..., H_{nh}\} \), the sub-sets of objects assigned to software \( S = \{S_1, ..., S_{ns}\} \), the set of constraints \( Cons \) and the set of requirements \( Req \) to return a value that measures the solution quality. The iterative partition algorithm is defined by the function

\[
\text{PartAlg}(H, S, Cons, Req, F_{\text{cost}}) \quad (1)
\]

returning \( H' \) and \( S' \) that verifies

\[
F_{\text{cost}}(H', S', Cons, Req) \leq F_{\text{cost}}(H, S, Cons, Req) \quad (2)
\]

when the applied cost function returns the minimum value under the best partition solution circumstances.

The value generated by the cost function is obtained from estimated metrics, related to the system constraints and requirements.

To execute the partial tasks needed by the partition process, the modules identified in figure 3 were used. Beyond the module that performs the conversion between the models used externally and internally by the partition process, the developed partition methodology includes partition algorithms (constructive and iterative), evaluation functions (closeness and cost) and metrics estimators. The following sections describe the modelling that is relevant for partition and the partition itself, with emphasis on the applied algorithms and evaluations functions and briefly presenting the metrics estimation.

### 2.1 System modelling

In related approaches, the uncommitted systems are commonly modelled with meta-models such as CDFG [14] [15], DFG [16], FSM [17], Petri net [18], CSP [19], an extended version of a previous meta-model [20], or a combination of these meta-models [6]. In spite of the meta-model diversity, most approaches transform the uncommitted system model into a flow diagram representation. The type of objects...
handled during the partition process is constrained by the selected meta-model, and the several approaches may present quite a different granularity, as a consequence of using distinct meta-models.

The PSM meta-model was selected to describe the systems at the partition process interface, which combines an HLL/HDL meta-model with HCFSM\(^2\) [7]. PSM adequately supports complex embedded systems modelling, since it includes the best features from both meta-models: behavioural hierarchy, concurrency, state definition, support to handle algorithmic and data complexity, behaviour completion, possibility of including exception handling and a graphical representation. Besides, modelling with PSM is a very intuitive task. The strongest limitations of PSM are the lack of structural hierarchy and automatic support to formally validate the models. In the present approach, the VHDL language was selected to describe variables and leaf program-states. VHDL allows an explicit and elegant modelling of communication and synchronization among concurrent activities.

A PSM model is described by an hierarchical set of program-states, where a program-state represents a computation unit that at a given time can be active or inactive. A PSM model may include composite or leaf program-states. A composite program-state is defined by a set of concurrent or sequential program-substates, and a leaf program-state is defined by a block of code on the chosen programming language. If the program-substates are concurrent they are all active at the same time; if they are sequential, just one program-substate can be active at a certain time.

On a composite program-state, the order by which sequential program-substates get active is determined by the directed arcs connecting them. There are two type of directed arcs: arcs that represent a transition when the substate activity is terminated

\(^2\)Hierarchical Concurrent Finite State Machine.
and simultaneously the condition associated with the arc becomes true, and arcs that
represent a transition immediately after the condition associated with the arc becomes
true. A transition on a directed arc means that the target substate will become active.

To represent a PSM model textual and graphic notations can be used.

**Internal representation**

To describe systems during the partition process a CFG type meta-model was devel-
oped: the PSM flow graph or simply PSMfg. The most relevant requirement of the
internal representation, not included on the PSM meta-model requirement list, is the
possibility of associating the information generated during the partition process with
the system model objects.

The motivations that lead to the development of a new meta-model were the need
to automate the partition process and the availability of a library with graphic and
computational support to edit graphs - LEDA[^3] [21]. By means of a set of adaptations
applied to the editor of generic directed graphs and the associated data structure, it
was possible to obtain the computational support to operate on PSMfg graphs. The
goals to achieve with the performed adaptations were: (i) to customize the graphic
characteristics of the nodes, generating the set of node types that will be presented
ahead; (ii) to increase the nodes and edges functionality, in agreement with its type;
and (iii) to introduce constraints on the interconnection between the different types
of node.

A PSMfg model is an acyclic, directed and polar graph, represen-
ted by a \( G = \{ V, E \} \) data structure that includes the list of nodes \( V \) and the list of edges \( E \). The
graph is acyclic when no paths on the graph are closed, it is directed because each
edge has a single direction and it is polar because it includes two nodes, one to enter
and the other to exit from the graph, from which all other nodes are successors and
predecessors, respectively [22].

The meta-model of the PSMfg represents the semantic of the PSM meta-model
and all the information needed by the partition process, such as the metrics es-
timate and the assignment of objects to partitions. To control the granularity of
the objects handled during the partition process, the PSMfg graph must be able
to represent the program-states structure. Since the program-states functionality is
described with VHDL, the PSMfg graph supports the following constructs of the
VHDL language: the parallelism associated with processes, the conditional con-
structs (if ... elsif and case), the cycles (while and for) and the con-
structs that suspend processes (wait).

The nodes of a PSMfg graph represent the variables and the program-states of a
PSM model, with the same or a thinner granularity, and they have associated with
them information that is relevant to the partition process, namely:

- which partition the graph node was assigned;

[^3]: Library of Efficient Data types and Algorithms.
which partitions the designer establish as being forbidden for this node;

- the required information to estimate the area occupied by the hardware and the system performance, which refers to metrics like the functional units, the storage elements or the interconnection elements area, the variables read/written by the program-state associated with the node, the computation time, the time spend on communication with others program-states or the execution frequency.

The different node types the PSMfg meta-model uses are those that: (i) define the entry/exit point of the system graph; (ii) indicate where the (parallel) processes begin/end; (iii) define the begin/end of a conditional construct; (iv) represent the control part of a cycle; (v) force a waiting cycle; (vi) assert one or more signals necessary to a waiting cycle; (vii) represent a variable; and (viii) do not fit in any of the previous types.

The edges, representing the control flow between nodes, have associated a branch probability (relative to the source node) and a label.

2.2 System Partitioning

In the present work, partitioning is a two-step process: (i) compute an initial partition solution with a constructive algorithm and (ii) successively improve it with an iterative partition algorithm.

A constructive algorithm

The analysis of several constructive partition algorithms revealed that: (i) the application of an exhaustive algorithm is not feasible since it demands an unacceptable computation time; (ii) the cluster growth and hierarchical clustering algorithms create the partition solutions in distinct ways, but produce identical results; (iii) the ILP methods generate optima solutions, do not require the application of an iterative optimization algorithm, but they demand a very high computation time and its formulation is hard to achieve; and (iv) PACE [23] and GCLP [3] algorithms, being strongly specific, are not attractive to be adapted to the present work. Since the solutions generated by the constructive algorithm feed the iterative improving process, its quality can be kept in a lower value. Thus, it was selected a constructive algorithm with a light implementation, the cluster growth (CG) algorithm. Although the optimization heuristic of the CG algorithm is quite simple, the capacity to generate solutions with quality is determined by the selected closeness function.

The process of creating a solution begins with the selection of the seed object for each partition. To select the partitions seed object, 4 methods were implemented: (i) random selection, (ii) manual assignment, (iii) combination of random selection

\footnote{Integer Linear Programming.}
with manual assignment and (iv) selection based on the communication among partitions. The manual assignment can be used to avoid that objects are assigned to an implementation for which they are clearly bad candidates. Having selected the seed object for each partition, the cluster growth algorithm assigns the remaining objects to the best possible partition. The best partition is chosen by the closeness function defined in equation 5 of section 2.3.

An iterative algorithm

Simulated annealing [20] [18] [24] [25] is among the most commonly used iterative partition algorithms, but it is also frequent to use genetic evolution [8], implementations of the Kernighan/Lin algorithm [26] [27], tabu search [24] [25] and specific algorithms. The evaluation of these algorithms has shown that Kernighan/Lin algorithm has a limited capacity to avoid local minimum of the cost function, the simulated annealing algorithm presents a stronger potential than greedy and Kernighan/Lin algorithms to achieve optima solutions, but the computation time is very high, and the tabu search algorithm decreases the computation time bounding the search for partition solutions to the neighbourhood of these solutions. The genetic evolution algorithms reduce the design space more efficiently, but the capacity of convergence to the optimum partition solution is inferior. Having in mind that the primary goal of partitioning is to find partition solutions with quality, tabu search and simulated annealing were selected for the iterative process. A thorough study was carried out with tabu search algorithm, and the results are presented in this paper.

The tabu search method (TS) can be seen as an extension of the local search strategies, where a new solution is found on the neighbourhood of the present solution, applying a well defined set of rules [28] [29]. When the iteration \( n \) of the search process tries to minimize the cost function \( F_{\text{cost}}(P_n) \), the new solution \( P_{n+1} \) is selected from the neighbourhood \( V(P_n) \) of the present solution, applying an optimization criterium. In general, the criterium expresses the objective of selecting the best solution present on the neighbourhood. The neighbourhood of solution \( P_n \) can be defined by the set of all the alternatives that result from the application of a rule that modifies the characteristics or attributes of \( P_n \). On the hardware/software partition problem, the transition from the present solution to a solution on its neighbourhood occurs when at least one object is moved from its current partition to a target partition, ending in a new solution. It is frequent an hardware/software partition problem to evaluate a high number of partition alternatives, which means that to find a solution with quality it is necessary a computation time equally high. To avoid that all the alternatives present on the current solution neighbourhood are evaluated, it is implemented a list with candidate solutions; this way, only a partial neighbourhood of the current solution is evaluated.

Although the tabu search is a local search strategy that tries not to stop in local minima of \( F_{\text{cost}} \), its policy embodies other features. This strategy was named tabu search since in every iteration parts of the design space are forbidden, e. g., some so-
lutions are considered tabu. To reach this goal, the tabu search implements a flexible memory structure that supports several search strategies, like avoiding local minima. The flexible memory includes short term (STM), long term (LTM) and medium term components (MTM). The short term components are based on the history of most recently visited solutions, the long term components are based on the most frequent solutions and the medium term components are oriented to solutions with quality and influent solutions. Using this information it is possible (i) to diversify the search, in order to escape the local minima, (ii) to intensify the search, to reinforce the convergence for the absolute minimum and (iii) to avoid cycles during the search.

To avoid a cycle during the search, the last \( L \) visited solutions are saved on the tabu list. While a solution \( P_n \) is on the tabu list, it is forbidden. This way, the search will not return, at least during \( L \) iterations, to a visited solution. The size of the tabu list, or the tabu tenure, is determinant to the evolution of a search, since it influences the restrictions applied to the design space that can be searched. For this reason, the more restrictive is a tabu, the less must be its tenure. The performed experiments resulted in the following recommendation: the objects and moves tabu tenure must be 5 to 10\% of the number of objects on the system description.

The temporary exclusion of solutions does not result solely in advantages for the tabu search method. The disadvantages arise when high quality solutions, the goal of searching, are excluded from the search. To overtake the inconvenience caused by the high quality solutions exclusion, TS methods have a mechanism that allows to withdraw the tabu classification of a solution, assuming it may be a solution with quality. This mechanism is called aspiration criterion. They can be defined aspiration criteria by objective, by direction of search and by influence \[28\].

The tabu search algorithm iteratively tries to improve the provided partition solution, assembles all the components that participate on the search and controls its evolution. The implemented algorithm \[2\] is a modified version of the one discussed in \[29\]. Namely, it only searches a partial neighbourhood of the present solution, it has a richer set of evolution strategies to apply when there are no eligible solutions with quality, and it applies a more efficient improvement when none of the moves improves the cost of the present solution. Partial neighbourhood searching, decreases the computation time per iteration by a factor close to the number of partitions, but the design space exploitation is less complete. Since the partial neighbourhood contains the best solutions, it is introduced an intensification element on the search.

The tabu search algorithm runs until a predefined number of iterations is reached and each search runs while a predefined number of iterations without improving the best solution is not exceeded. The number of iterations that can be executed without improving the partition solution should not be neither too high - to avoid wasting iterations around a local minimum - nor too low - to increase the possibility of converging to a local (or absolute) minimum of the cost function.

On every iteration of the search process, the partial neighbourhood of the present solution is analysed and the move to be executed can be selected from one of the
following ordered alternatives:

1. The move that generates the largest improvement on the partition solution cost and that obeys one of the following conditions: it is not tabu or it is tabu but can be executed due to an aspiration criterium;

2. The move that is not tabu and leads to the smallest increase on the partition solution cost; the cost of the solutions that result from the moves is decreased by the application of a negative improvement;

3. The “least” tabu move, the least frequent move, the move that in the past resulted in the best cost variation or the move of the object that stays longer in the same partition.

At the end of every iteration the performed move, the inverse move and the moved object are classified as being tabu, the history is updated with the information about the move and the moved object, the moves and objects tabu tenure, the best solution found and the number of iterations are updated and, at the end of a search, a new initial solution is generated.

Experimental results show that, at the beginning of each search, the information saved on the history of moves and moved objects must be reset. Otherwise, the capacity of converge to the optimum solution is reduced, since the improvement used on the second move alternative, proportional to the number of iterations an object is not moved, would regularly select every object.

Applying all types of tabu classification can be very restrictive to the search. A subset of tabu classifications was selected, which decreases the dimension of the neighbourhood to be searched, helps to avoid cycles and does not place excessive restrictions to the search. The following tabu classifications were selected: (i) move a given object from a source partition to a target partition; (ii) all moves of a given object; and (iii) the inverse (move) of the move that originated the present solution.

The implemented TS method includes two types of aspiration criterium: (i) by objective, when the first alternative selects a move with quality that is classified as being tabu; and (ii) a default criterium, when the third alternative selects the “least” tabu move.

The implemented memory structure registers the history of performed moves and the history of moved objects. For each performed move, the history of moves saves the source and target partitions, the tabu tenure (STM), the execution frequency (LTM) and the achieved cost variation (MTM); for each moved object, it saves the tabu tenure (STM), the frequency of move (LTM), the number of iterations an object remains on the same partition (LTM) and the achieved cost variation (MTM).

It was implemented a neighbourhood with a simple structure since a neighbourhood with a complex structure would increase greatly the computation time. While on a partition problem with $nObj$ objects and $nPart$ partitions, the size of the simple neighbourhood is $nObj \times (nPart - 1)$, on a generic complex neighbourhood,
where each iteration executes a series of \( n \) Moves moves, the number of alternatives that make up the neighbourhood is defined by the equation 3. The value defined by this equations is much higher the number of alternatives on a simple neighbourhood.

A complex neighbourhood favours the diversification on the search, which means an increased capacity to avoid the local minimum but also an increased difficulty to converge to the optimum partition solution.

\[
\text{size} (V) = (nPart - 1)^{n \text{Moves}} \cdot \binom{nObj}{n \text{Moves}} = (nPart - 1)^{n \text{Moves}} \cdot \frac{nObj!}{n \text{Moves}! \cdot (nObj - n \text{Moves})!} \tag{3}
\]

The partial neighbourhood, or the list of candidate solutions, considered on every iteration of the TS algorithm is a subset of the present solution neighbourhood, with a size that remains fixed during all the search process and equals the number of system objects. The \((nPart - 1)\) moves per object that define the neighbourhood were decreased to only one move per object on the partial neighbourhood, decreasing the computation time by a factor close to the number of partitions. The subset of moves that define the partial neighbourhood is made up by the best move for each object of the system description. The best moves are computed by a function identical to the closeness function of the cluster growth algorithm \(F_{prox}\) on equation 5.

Part of the TS algorithm potential is consequence of executing several searches, each one with a different initial solution. The method used to generate the initial solution of the searches combines two strategies: intensification - the new initial solution results from the best evaluated partition solution - and diversification - the assignment of a significant percentage of objects is modified, according to the long term memory. The rule is to execute the least frequent moves, but after a number of searches without improving the best solution, the choice can be to move the least frequently moved objects to a randomly selected partition. The random selection reinforces the diversification on the search. Given that the percentage of moved objects is a parameter of the algorithm, it is possible to control the relation between the intensification and the diversification applied on the generation of a new initial solution. The suggested value for the percentage of objects to be moved is 20%.

The implemented algorithm includes the following intensification elements:

- to create the list of candidate solutions with the highest quality solutions present on the current solution neighbourhood;
- to create the initial solution for a new search based on the best evaluated solution;
- to select, for third evolution alternative of the TS algorithm, the move that in past resulted in the best cost variation;

and the following diversification elements:
to apply, on the second evolution alternative of the TS algorithm, a cost improvement based on the number of iterations the objects remained in the last partition $P_k$ they were assigned ($NIMP_k$); this improvement, described by equation 4, strongly favours the move of the objects that are not moved regularly, since they have a high $NIMP$; thus, the search is directed to less explored zones and a diversification component is introduced on the search:

$$improvement(P_k) = \frac{NIMP_k}{nObj}$$  \hspace{1cm} (4)

- to create the initial solution of a new search moving a percentage of the least frequently moved objects or a percentage of objects selected randomly (after a number of searches);
- to select, as the third evolution alternative of the TS algorithm, the least frequent move or move the object that remains longer on the same partition.

2.3 Evaluation functions

This section describes the evaluation functions (closeness and cost) that guide the partition algorithms (constructive and iterative) on the creation and improvement of partition solutions.

Closeness function

The best partition used to assign the objects, on every iteration of the constructive partition process, is chosen by the closeness function defined in equation 5.

$$F_{prox} = f\left[\frac{F_{var}(M_{com1})}{F_{psHwSw}(M_{cmp1}, M_{cmp2}, M_{com2})}, \frac{F_{psHwSw}(M_{area}, M_{com2})}{F_{psHw}(M_{area}, M_{com2})}\right]$$  \hspace{1cm} (5)

where $M_{com1}$ ($M_{com2}$) represents the communication intensity among a variable (program-state) and the program-states (variables) assigned to the partition, $M_{cmp1}$ ($M_{cmp2}$) is the software (hardware) computation time of a program-state and $M_{area}$ is the area occupied by all the variables and program-states assigned to the partition. The $F_{var}$ function is used on variables assignment and the $F_{psHwSw}$ and $F_{psHw}$ functions are used on the program-states assignment. On every moment of the constructive process, the $F_{prox}$ function measures the closeness among the object to be assigned and the objects previously assigned to each partition.

If the object to be assigned is a variable that is a bad candidate to hardware, meaning that the area it occupies in hardware exceeds a defined limit, the $F_{var}$ function suggests an assignment to software. If the variable is not a bad candidate to hardware, it is assigned to the partition that presents the higher communication intensity with this variable, e.g., to the partition $p$ that presents the best $M_{com1}[p]$ value.
When a program-state is being assigned, if the $F_{psHw;sw}$ function indicates that software is the best partition to assign it, the program-state is immediately assigned to software. Otherwise the best hardware partition is selected by $F_{psHw}$, a function that is more appropriated to distinguish the assignment to the different hardware partitions.

For example, the metric $M_{com1}$ used to select the best partition $p$ to assign a variable $v$, is computed with equation 6. The communication intensity $M_{com1}[p]$ simply measures the number of times the variable $v$ is read/written by the program-states assigned to the partition $p$.

$$M_{com1}[p] = \sum_{o \in (rdO(v) \cap p)} rdV(o).nRd(v) \ast rdV(o).pRd(v) \ast FN(o) + \sum_{o \in (wrO(v) \cap p)} wrV(o).nWr(v) \ast wrV(o).pWr(v) \ast FN(o)$$

\[ (6) \]

where

- $rdO(v)$ ($wrO(v)$) is the set of program-states that read (write) the variable $v$;
- $rdV(o)$ ($wrV(o)$) represents the set of variables read (written) by the program-state $o$;
- $rdV(o).nRd(v)$ ($wrV(o).nWr(v)$) is the number of times the variable $v$ is read (written) by $o$ on every execution;
- $rdV(o).pRd(v)$ ($wrV(o).pWr(v)$) is the probability of variable $v$ to be read (written) by $o$;
- $FN(o)$ is the execution frequency of $o$.

**Cost function**

The cost function applied on the iterative partition process considers as being optimum a partition solution that respects the target architecture constraints and achieves the design requirements, as opposed to considering as being optimum a solution that uses the least hardware area and/or achieves the best performance. To reach this goal the function includes a term, per constraint or requirement, whose value is proportional to the degree this constraint or requirement is not observed on the partition alternative (equation 7).

$$F_{cost}(H,S,Cons,Req) = \sum_{i=1}^{n} K_i \ast f_i(M_i, C_i)$$

\[ (7) \]

where

- $H$ ($S$) is the set of hardware (software) partitions;
\( C_{\text{Cons}} = \{ C_1, C_2 \} \) is the set of design constraints, with \( C_1 \) being the constraint applied to the area of the hardware partitions data path \( (M_1) \) and \( C_2 \) the constraint applied to the area of the respective control unit \( (M_2) \);

\( Req = C_3 \) is the performance required from the system \( (M_3) \);

\( M \) is the set of metrics \( M_i \), whose constraints \( Cons \) and requirement \( Req \) apply to;

\( K_i \) is the coefficient applied to the metric \( M_i \);

\( f_i(M_i, C_i) \) represents the contribution of the metric \( M_i \) to the cost function and it is defined by equation 8.

\[
f_i(M_i, C_i) = \left\{ \begin{array}{ll}
\sum_{P_j \in H} \text{MAX} \left[ \text{excess}(M_i(P_j), C_i), 0 \right], & i = 1, 2 \\
\text{MAX} \left[ \text{excess}(M_3, C_3), 0 \right], & i = 3
\end{array} \right.
\]

(8)

where

\( M_i(P_j) \) is the value of metric \( M_i \) for the hardware partition \( P_j \);

\( C_i(P_j) \), the value of the design constraint \( C_i \) applied to the hardware partition \( P_j \), was replaced by \( C_i \) on equation 8; on the considered target architecture, the pairs (FPGA,CPLD) that implement the pair (DP,CU) of the hardware partitions include the same devices;

the term \( \text{excess}(m, c) \) is given by

\[
\text{excess}(m, c) = \frac{m - c}{c}
\]

(9)

The estimates for the area \( (M_1 \) and \( M_2 \) are computed by partition, while the estimate for performance \( (M_3) \) is relative to all the system.

### 2.4 Metrics estimation

Metrics estimation aims to compare partition alternatives, which requires a high degree of fidelity rather than a high accuracy. However, it is expected that a high accuracy corresponds to an equally high degree of fidelity.

The estimation operates on the system graph, modelled with PSMfg, considers an hardware model (with data path and control unit), a software model (with a predefined set of instruction types) and a communication model (for inter-partition communications). The code optimization performed by the compiler - related to pipelining, superscalarity and memory hierarchy - is measured as a factor obtained by simulation. This procedure is acceptable on most partition problems applied to embedded
systems. One difference to a significant part of the approaches, is the emphasis given to the estimation of metrics related to inter-partitions communications.

To obtain accurate estimates, detailed models for the used resources were developed, especially the hardware and communication models, and the estimation runs in two abstraction levels: program-state and system. The incremental update of the estimates and the estimation in two levels both help to decrease the computation time.

Low level estimates, which are used by the system level estimates, are computed at the program-state abstraction level, the computations are performed once per partition session and the estimates are more accurate. Estimates for metrics relative to the system objects are computed at the program-state level. Examples of these metrics are the software and the hardware computation times, the area occupied by functional units, multiplexers and variables, the read/written variables and the program-states that read/write variables. To obtain these estimates, low level metrics are required: these include the execution time of the arithmetic/logic operators and the area occupied by multiplexers, arithmetic/logic operators and memory elements.

At the system level, metrics are estimated at a higher level and the computations are repeated on every iteration of the partition process. The estimates are less accurate and, whenever possible, the estimates are simply updated. The metrics estimated at the system level are the system performance and the area occupied by the data path and the control unit of the hardware partitions. System performance is computed through explicit scheduling at the state-program level. By ignoring the scheduling at the system level, the computation time is decreased and the obtained performance tend to be over estimated.

The computation of the execution times follows a simple software model, that estimates computation time by type of executed instruction (the built prototype follows the IA-32 architecture model) and considers the optimizations performed by the compilers as a factor obtained by simulation.

The developed hardware model focus on the area of a partition, which includes the area of the data path - the functional units, the storage elements, the interconnection resources and the resources of the interface with other partitions - and the area of the control unit - the area of the state machine associated with the partition data path, which includes the state register, the output logic and the next state logic. Experimental results confirmed the state register as the dominant term on the area of the state machine, ranging from 60 to 80%.

The developed communication model defines the timings and the resources associated with the communication between partitions. The model supports the register access communication mechanism, by polling and by interruption. At the beginning of every search of the iterative partition process, estimates for communication times are computed. These estimates will be updated whenever an object is moved from one partition to another one, but only for the moved object and/or those objects that communicate with the moved objects.
3 Validation of the partition methodology

The proposed methodology was validated on a CPU-based architecture coupled to a reconfigurable board (briefly described below), through two case studies that represent data flow dominated embedded systems: one clearly suggesting a software implementation, while the other is oriented for an hardware implementation.

A quantitative evaluation compared automatically generated solutions with manually optimised hardware/software implementations, looking into two main results: the quality of the partition solutions (measured by feasibility and performance) and the quality of the estimates (measured by accuracy/fidelity). The methodology can be further evaluated by its performance, e. g., by the computation time needed to generate the partition solutions and the support to implement these solutions, namely to synthesize the interface between partitions.

3.1 Prototype system

The prototype system applied on the partition methodology validation includes a target architecture and a partitioning tool.

The considered target architecture contains a reconfigurable platform (EDgAR-2) and its host system. The EDgAR-2 board is an FPGA/CPLD based system, with a PCI interface and fully in system programmable (ISP) [1] [30]. The board structure, shown in figure 4, contains an array of 4 pairs (control unit, data path), called processor modules (PMs), which are 2-way interconnected with dedicated buses, forming a PM pipeline; they are also connected to a different set of 8 lines in the 32 bits PCI data bus. FPGAs implement the data paths, while CPLDs are better suited to implement the control units.

The EDgAR-2 architecture was designed to directly accommodate a finite state machine with data path (FSMD) model. Since the architecture implements several concurrent FSMDs, it is suitable to map descriptions modelled with concurrent FSMDs (CFSMD) [31], hierarchical concurrent FSMD (HCFSMD) or program state machine (PSM) meta-models [32].

Although EDgAR-2 may not be considered a typical reconfigurable board - it is composed of both FPGAs and CPLDs, and it lacks on board RAM - it is particularly adequate to validate a general purpose hardware/software partition methodology due to these extra challenges.

The applied tool was parTiTool, a framework based on the LEDA library [21] and which allows the visualisation, edition and partitioning of PSMfg graphs. This framework also includes support to detect errors on the graphs structure and to visualise the output of the partitioning process. The major part of the operations needed by the graphs visualisation and edition is supported by the classes GRAPH and GraphWin of the LEDA library.
3.2 Case studies

The partition methodology was validated with a detailed analysis of two case studies: the application of a Sobel filter to an image (convolution) and the DES\(^5\) cryptography algorithm [33]; the first one is oriented for a software implementation and the latter suggests an hardware implementation.

The application of a Sobel filter \(F\) (with \(X\) by \(Y\) pixels) to an image \(I\), runs through two steps: (i) for every pixel \((j, i)\) of the original image \(I\), which colour is \(I(j, i)\), an area with the filter size and centered on pixel \((j, i)\) is convoluted with the filter \(F\), generating a new value \(If(j, i)\) for pixel \((j, i)\) (equation 10); (ii) with the minimum and maximum of the filtered image \(If\), \(m(If)\) and \(M(If)\) respectively, the filtered image is normalized to the colour range of the original image \(r(I)\), generating the filtered and normalized image \(In\) (equation 11).

\[
If(j, i) = \sum_{k=0}^{Y-1} \sum_{l=0}^{X-1} I(j - \lfloor \frac{X}{2} \rfloor + l, i - \lfloor \frac{Y}{2} \rfloor + k) \ast F(l, k) \tag{10}
\]

\[
In(j, i) = \frac{r(I)}{M(If) - m(If)} \ast [If(j, i) - m(If)] \tag{11}
\]

The implemented DES algorithm applies a set of transformations to the input data (sample), which depend on these data and on the secret key. This key is also altered.

---

\(^5\)Data Encryption Standard.
during the different iterations of the encrypt process. Every sample to encrypt goes through an initial permutation \( IP \), a set of transformations that depend on the secret key and a final permutation \( FP \), inverse of \( IP \) (figure 5). The set of transformations that depend on the secret key is defined by an encryption function \( f \) and a key scheduling function \( KS \).

The function \( f \) includes the expansion \( E \), the substitution tables \( S-box \) and the permutation \( P \). The information generated by the initial permutation \( IP \) is splitted in two 32 bits halves: the least significant part \((R)\) feeds function \( f \) and the most significant part \((L)\) is the input for an exclusive-OR operator. At the end of a round, the two halves of the sample to encrypt are swapped and the round is repeated. The algorithm evolves in 16 rounds, in order to “circulate” the sample to be encrypted.

![Figure 5: Block diagram of the DES algorithm.](image)

The key scheduling \( KS \) generates a 48 bits key for each of the 16 rounds of the DES algorithm, through a linear combination of the 56 bits secret key. The \( KS \) module includes a permutation \( PC1 \), a register, a permutation \( PC2 \) and a shift left (right) operator, applied on the encrypt (decrypt) process.

The dimension of the partition problem associated with both examples and the parameters used on the resolution with the tabu search algorithm are synthesized on table 1. The high number of objects indicated for both examples is a consequence of using explicit parallelism at the system description.

### 3.3 Experimental results

The best partition solution, generated by tabu search for the DES example, assigns program-states and variables to partitions \((SW \text{ or } HW1 \text{ to } HW4)\) as it is illustrated in figure 6. The objects in the upper part of the figure represent PSM variables and the remaining objects are the PSM program-states equivalents.

When the automatic partition solutions are compared with manually optimized hardware/software implementations, the measured performance of the best automatic partition solution reached 72 to 92% of the manual implementation performance, being superior on the cryptography example. These results can be improved by detailing the estimation models and by tuning the granularity of system model objects, which
**Table 1: Parameters used on the partition process with the tabu search algorithm.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Convolution</th>
<th>Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N^o ) partitions</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>( N^o ) objects</td>
<td>217</td>
<td>372</td>
</tr>
<tr>
<td>( N^o ) iterations</td>
<td>43400</td>
<td>74400</td>
</tr>
<tr>
<td>nBest</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>pMoves</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>nRand</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>( TT_{move} )</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>( TT_{Move} )</td>
<td>18</td>
<td>22</td>
</tr>
<tr>
<td>( TT_{obj} )</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>

\( nBest \) - Number of iterations since the best partition solution was found.

\( pMoves \) - Percentage of objects to be moved when the initial solution of a new search is created.

\( nRand \) - Number of searches without improving the best partition solution in order to execute “random” moves when creating the initial solution of the next search.

\( TT_{move} \) - Moves tabu tenure.

\( TT_{Move} \) - Inverse moves tabu tenure.

\( TT_{obj} \) - Objects tabu tenure.

will significantly increase the computation time. The different experiments done with the mentioned examples always ended in feasible partition solutions, e.g., solutions that respect the target architecture constraints, a proof that the applied closeness and cost functions correctly control the partition process.

The accuracy and fidelity of the estimates for the performance and for the area occupied in hardware were also evaluated. The accuracy of the system performance estimates ranged from 82 to 98\%, being higher on the cryptography example due to its lower complexity. A fidelity ranging from 83 to 100\%, almost coincident with the accuracy range, suggests that the computed estimates are reliable. The accuracy of the estimates for the area occupied by the hardware partitions data path was 92 to 99\%, being identical on both examples. The accuracy of the estimates for the area occupied by the hardware partitions control unit ranged from 89 to 96\%, with very close results on both examples. The obtained results show that the control unit area depends mainly on the state register area, that in turn is proportional to the number of states. For the whole set of metrics and examples, the accuracy and fidelity of the estimates were always above 82\%, a very rewarding result. The results obtained with the partition process are summarised on table 2.

When it was decided to compute accurate estimates, the performance of the partition methodology tool ended penalized. One way of improving the tool performance is to optimize the estimation of the system execution time. The time complexity \( O(nObj) \), expected for the tabu search algorithm, was experimentally proved. Since
Figure 6: PSMfg model illustrating the best partition solution from the TS algorithm.

The computation time varies linearly with the number of objects on the system description, on large sized systems the time required to find the best partition solution is high. However, in the majority of cases, the first searches of the partition process generate a solution with quality.

The support given by the partition methodology to the implementation of the systems was also evaluated. The automatic synthesis of the interface between partitions is a straightforward implementation that uses the data from the estimation of the area occupied by the resources of the interface between partitions and the communication time between partitions.

76
### 4 Conclusions

The cluster growth constructive algorithm follows a straight optimization heuristic, which proved to be able to generate solutions with quality, when guided by an adequate closeness function.

The results from the performed experiments with tabu search (TS) algorithm recommend that objects and moves tabu tenure must be 5 to 10% of the number of objects on the system description. To decrease the computation time while keeping the capacity to generate solutions with quality, the implemented TS algorithm only searches a partial neighbourhood, has a richer set of evolution strategies, applies a more efficient improvement and includes a richer set of diversification/intensification elements. To avoid the reduction of the capacity of converging to the optimum solution, the history of moves and moved objects must be reset by TS at the beginning of each search. A subset of tabu classifications was selected, which decreases the computation time, helps to avoid cycles and does not place excessive restrictions to the search. A neighbourhood with a simple structure also helps to decrease the computation time. The goal of the cost function applied by TS is to achieve the best partition solution with the available resources.

To generate accurate estimates, while keeping the computation time as low as possible, the implemented estimation methodology (i) uses detailed models for the hardware resources, (ii) runs in two abstraction levels and (iii) uses incremental updating.

The obtained results show that the best automatic solution from the TS algorithm achieves 72 to 92% of the manual partition solution performance. This is an interesting result limited by (i) the optimizations introduced on the manual solution implementation, (ii) the simple software estimation model and (iii) the fine granularity used with the objects. The different experiments always ended on feasible partition solutions, which proves that the partition process is adequately controlled by the evaluation functions.

The accuracy of the performance estimates, the area of the data path and the area of the control unit estimates, was respectively 82 to 98%, 92 to 99% and 89 to 96%. The estimates accuracy obtained with both examples, DES and convolution, was very close. This consistence on the accuracy suggests a reliable estimation. For all metrics and examples, the accuracy and fidelity of the estimates was always above 82%, an

![Table 2: Results obtained with the partition process.](image-url)
interesting result that in many cases overcomes the published results.

The time complexity $O(n)$, foreseen for the implemented TS algorithm, was confirmed on the experiments performed with parTiTool. The time necessary to compute the best partition solution is high, but in most cases 10% of this time is sufficient to find a solution that achieves a performance close to 90% of the best solution.

The estimated data for the interface resources and the communication time, simplifies the automatic synthesis of interfaces.

Some directions are being considered for future work: (i) evaluation of the methodology with more and differentiated case studies, namely more complex and control dominated systems must be tested; (ii) integration of the methodology on a broader one, which is used to develop concurrent systems that are implemented on a parallel, distributed and heterogeneous architecture; (iii) implementation of other iterative algorithms – beyond TS and SA – where different optimization strategies may lead to better results with some examples, to increase the partition success; (iv) optimization of the system performance estimation, to improve the performance of the partition methodology, strongly dependent on the time needed to estimate this metric.

References


An Approach to Quality Estimation in Model-Based Development

Jens Peter Holmegaard, Anders Peter Ravn, and Peter Koch

Center for Embedded Software Systems
Aalborg University, 9220 Aalborg, Denmark
jpho@kom.aau.dk, apr@cs.aau.dk, pk@kom.aau.dk

Abstract
We present an approach to estimation of parameters for design space exploration in Model-Based Development, where synthesis of a system is done in two stages. Component qualities like space, execution time or power consumption are defined in a repository by platform dependent values. Connectors are treated as special components; they have platform dependent overhead values for the qualities and composition functions, defining how qualities are computed from the values of connected components. The approach is exemplified with a prototype estimation tool applied to an OFDM-decoding module modelled in SystemC.

Keywords: Model-Based Development, Embedded Systems, Quality Estimation.

1 Introduction

Hardware/software codesign is based on the assumption that there are benefits to be gained by partitioning a system into components in different technologies. A key issue therefore has been to estimate non-functional properties or qualities of a given partition. A similar issue occurs when an embedded system is produced using Component-Based Development [5], which is based on intellectual property (IP) reuse. The partition is here given by the components with their interfaces, which are glued together using standard connectors. An example of this trend is found in the language SystemC, which extends C++ with modules incorporating interfaces based on CSP-like communication. A system is built from components existing in-house or available from external vendors of IP. Components are chosen in order to satisfy the functional requirements of the system to be built and are assembled into an implementation. Components are deployed in a framework, which provides the necessary run-time environment for the components and their interaction as provided by a platform and middleware. In this context, we speak of Model-Based Development, when
a system is designed on the basis of interface specifications (models) for the components, followed by a synthesis process that assembles concrete components for a specific platform with a specific interconnection framework. Assuming the presence of appropriate methodology and tools support, Model-Based Development with components reduces development time and in turn development expenses.

However, embedded systems are often subject to requirements of extra-functional character, such as latency, throughput, code size, chip area, etc. Often, as in codesign, a trade-off has to be made between such parameters and cost, because the cost of the completed system cannot be ignored. Therefore one needs to check the varying qualities of a number of functionally equivalent system implementations, just as one would do with codesign partitions. In the case of Model-Based Development, analysis of qualities is more complex than in the case of codesign, where one typically will analyse a component at the very operational level of C-code. Furthermore, redesign and simulation studies of the completed systems are expensive, because the reason for introducing Model-Based Development is to develop more complex functionalities. Thus we want to explore approaches to assessment of qualities of possible implementations at an early stage of the design cycle. The particular approach we report in this paper is use of model-based estimation tools, facilitating design space exploration of potential implementations at an early stage.

1.1 Related Work

Due to the importance of estimation and design space exploration, these problems have been subject of extensive research within hardware/software codesign and system-level synthesis. Area and delay estimators for design space exploration in the context of FPGA synthesis with the MATCH compiler are presented in [1]. Input to the compiler is an application description in MATLAB. Area estimates are quantified as number of required CLBs and are based on scheduling/lifetime analysis of operators and registers. Delay estimation is based on the delay of the RT level components used, taking into account number of inputs and input word length. Thus, estimation is based on component properties, but at a low level of abstraction. An approach addressing estimation of execution time and program memory size of software mapped to a specific processor is presented in [4], [6]. The approach is based on converting the instruction-set of the processors of interest into a generic instruction-set model, annotated with cycle count and code size from the original instruction-set. Now a model algorithm can be analysed, for example by summing up instruction execution times or code sizes. The approach actually views instructions as elementary components of the program. In [2], the Design Trotter framework for design space exploration is presented. Based on a hierarchical control-data flow graph, metrics for guidance in the design space are proposed. These metrics quantify memory, control, and processing orientation as well as parallelism of the graph, and are used to guide the choice of implementation platform.

These results show that faithful estimation is possible. But they work with low
level components and are specific for an underlying implementation platform. In our work we raise the level of abstraction; components correspond to embedded software or hardware components at an algorithm-level of abstraction. Moreover, to facilitate a model-based approach, the platform is represented by parameterizing with platform specific attributes.

1.2 Overview

In Section 2 we discuss Component-Based Development. Components and synthesis as well as design space exploration are addressed. Section 3 discusses Model-Based Development and estimation. Then, in Section 4, we present the structure and implementation details of an estimation tool for predicting properties of component assemblies with respect to the deployment platform. The section presents also a worked example, before we reach conclusions and ideas for future work in Section 6.

2 Component-Based Development

A component is a unit of functionality. For interaction with the surroundings it has an interface describing input and output ports of the component. Component interaction semantics vary. In some frameworks, the ports use synchronous access corresponding to method calls, whereas in other frameworks, they support asynchronous message passing.

In Component-Based Development a system is built by composing selected components. A system is essentially a (composite) component, i.e., it provides some functionality over an interface. Hence, it may serve as a component (a sub-system), when building a larger system.

The interface hides the internals of the component from the environment. This encapsulation of functionality and the corresponding decoupling from communication is an important prerequisite for re-usability, known as orthogonalization of concerns [14].

In order to work, a component requires an execution engine. This is provided by the platform corresponding to the technology of the component: a certain processor, FPGA technology, etc. Hence, concrete components are platform specific. The platform for a system may be either a single processor or a multiprocessor, homogeneous or heterogeneous, since the execution of the components may be distributed. Thus, a system may include both software and hardware components. Apart from the technology platform, hardware components are just like software components in the context of system development.

The components forming a system communicate and synchronize through mechanisms, provided by middleware, for instance CORBA ORB. The middleware is required by the component system and hence necessary regardless the complexity of the platform. The concrete implementations of the mechanisms provided by the mid-
System

Platform

Middleware

Figure 1: Layers of a component-based implementation.

Figure 2: The component-based synthesis process.

dleware can be viewed as specialized components and hence conform to the same characteristics as discussed above. The middleware and the platform together constitute the execution environment of the component system. This is the framework in which the component system is deployed [5]. Together, the component system and corresponding deployment framework constitute a complete implementation as illustrated in conceptual form in Fig. 1.

2.1 Synthesis

The process of building a component-based system is illustrated in Fig. 2. The process may be accomplished manually, semiautomatic or fully automatic. Taking the specification as input, a system is built by connecting components from the component repository and adding the middleware components required for communication and synchronization. The specification represents an abstract description of the system to be built, defining the properties to be satisfied by the system.

Formally, synthesis is a relation, $S$, such that

$$ S : S \times C \leftrightarrow C $$

where $S$ denotes possible specifications, and $C$ represents a set of platform specific components. The result of synthesis is also a collection of components satisfying the functional properties of the specification and targeted at one or more platforms. As indicated, the components may be organized in component libraries, $\{C_0, C_1, \ldots \}$. 
where the indices refer to platforms, and the set $C$ is thus the union of such libraries. If a library organization is used, the synthesis relation shall place the resulting components in appropriate libraries.

Synthesis is a relation since it may generate a number of possible implementations for the same specification, $s \in S$, combining a given set of platform specific elementary components, $C \in C$, in various ways. This is the set, $\Omega$, given as

$$\Omega = S(s, C) = \{c_1, c_2, \ldots\}$$

(2)

Each system obtained in this way by a synthesis, $S$, according to Equation 1 will have unique qualities since they are built from different elementary components corresponding to specific platforms. It is part of the synthesis process to select components such that they are compatible with respect to platforms, cf. the previous remark about organizing $C$ as libraries.

### 2.2 Design Space Exploration

To obtain the qualities of an implementation, we postulate a quality evaluation function, $q$, to be

$$q : C \rightarrow Q$$

(3)

Recall that $C$, the set of components, include the set of possible system implementations. The range of the quality evaluation function, $Q$, is a set of $m$-tuples, $Q = Q_1 \times Q_2 \times \cdots \times Q_m$, where $Q_i$ is a specific quality. A quality, $Q_i$, is usually some totally ordered discrete or dense set, e.g., the set of real numbers from 0 to 1, or a bounded set of natural numbers. The attributes evaluated by the quality evaluation function form a design space. The number of qualities evaluated, i.e., the dimension of $Q$, determines the dimension of the design space. For instance, suppose two different qualities, $Q_1$ and $Q_2$, are evaluated, the design space will be two-dimensional. Extending the quality evaluation function, $q$, point wise to a set function, and using it

---

Figure 3: Design space in two dimensions.
on the set of systems built by a synthesis, $\Omega$ of Equation 2, we obtain a mapping onto the design space at points $q(\Omega) = \{(q_1, q_2)_{c_1}, (q_1, q_2)_{c_2}, \ldots\}$, according to Equation 3. Fig. 3 illustrates a two-dimensional design space with the mapping of three implementations evaluated after synthesis. In this example, the qualities are code size and cycle count. The points in the design space correspond to actual measurements for a Viterbi algorithm and various platforms in terms of processors from Analog Devices (AD), Texas Instruments (TI) and ARM, respectively. The design space of Fig. 3 illustrates the heavy influence of the platform on the qualities of the component.

The mapping of the set of components built from the elementary component library using a particular synthesis from a specification into a design space is referred to as design space exploration. The quality requirements of the specification define a subspace within which trade-offs among the qualities in the set $Q$ with respect to implementations can be explored and a component composition and corresponding platform can be chosen subject to some criteria.

3 Model-Based Development and Estimation

The straightforward approach to exploring the design space is to build all possible implementations, and measure the qualities of interest, using some quality evaluation function as discussed in the preceding section. This approach, however, is in most cases impractical considering i) the time it takes to synthesize the system implementations, and ii) the evaluation itself, i.e., measuring the qualities may be very time consuming. Hence, one must try to estimate the design space points without actually building the system implementations. This may be accomplished using Model-Based Development.

Fig. 4 illustrates a Model-Based Development process. First, a design model is generated by model synthesis. This synthesis requires only component models represented in the form of the component interfaces, since these define the rules of combination. If $\hat{C}$ represents the set of possible design models, the model synthesis is, as before in Equation 1, a relation, $\hat{S}$, such that

$$\hat{S} : S \times \hat{C} \leftrightarrow \hat{C} \quad (4)$$

A set of libraries of possible elementary component models represented by their interfaces, $\hat{C}$, is used for the synthesis. These interface libraries correspond to the libraries of elementary components discussed in the preceding section.

A model is only an intermediate step toward an implementation. In order to complete the synthesis, the model needs to be compiled. This step is also shown in Fig. 4. Given a compiler or translator $T$, we have

$$T : \hat{C} \times C \leftrightarrow C \quad (5)$$

where $C$ as previously said, provides the actual platform specific component implementations and interconnection implementations. The translation of Equation 5 is
a relation since a model may represent several implementations using various platforms.

In Model-Based Development, instead of measuring properties on the actual implementations, we seek to estimate the quality attributes of interest, using a quality estimation function, $\hat{q}$, applied to the design models, $\hat{C}$. The design model captures the structural information about the system, which will influence quality attributes and hence be required for estimation. Moreover, as the quality attributes of a system generated by the compilation function are obtained from the components constituting the assembly, the estimation function will need the quality attributes of the components as well in order to perform predictions. Introducing the set $A$ to represent platform specific libraries of qualities, in order not to confuse with $\hat{C}$, a quality estimation function, $\hat{q}$, is, cf., Equation 3

\[
\hat{q} : \hat{C} \times A \rightarrow Q
\]  

Note that the range of qualities is unchanged.

A useful $\hat{q}$ must provide good estimates. In practice, since the quality estimate is only going to be used to choose between a number of candidate system implementations to be compiled from the architecture model, it suffices for the estimation function to provide an estimate of good fidelity. Fidelity of an estimate, means that the likelihood that a decision based on the estimate turns out the same as a decision based on the actual measures [6]. Assuming a specific quality, $Q_i$, to be some totally ordered set as discussed in the preceding section, the fidelity requirement can be expressed as

\[
\hat{q}(a_1, \hat{C}_1) \leq \hat{q}(a_2, \hat{C}_2)
\]

\[
q(T(a_1, C_1)) \leq q(T(a_2, C_2))
\]

for models $a_1$ and $a_2$ in $\hat{C}$ and libraries $\hat{C}_1$ and $\hat{C}_2$ included in $\hat{C}$. In other words, fidelity means that the model estimate preserves the ordering of the implementations. If this requirement is satisfied, design space exploration can be advanced in time under design process, using attribute models of the elementary components to build design models and obtaining estimates using a model.
4 Estimation Tool

Having identified the requirements for performing estimation, we have developed a prototype implementation of an estimation tool for Model-Based Development, corresponding to the expression of Equation 6. Fig. 5 shows the structure of the estimation tool. The tool takes as input a system model representing the component composition and a repository storing the attributes of interest for the components of the composition. Moreover, attributes for the middleware are provided as input. Hence, the set of all attribute in Equation 6, is separated into two sets in our tool implementation; this was done for practical reasons. The model is passed through a lexical analysis and a parser in order to obtain an abstract syntax tree or parse tree. The parse tree reveals the structure of the assembly as well as the hierarchical composition of components. Traversing the parse tree, it is annotated with the attributes from the repository. Finally the analysis produces the estimate through another traversal, where the middleware overhead is computed based on the structural information of the model. For the prototype implementation we have chosen the Java programming language. To build the lexer/parser, we have used the Java Compiler Compiler (JavaCC) [9] and Java Tree Builder (JTB) [10] tools. Taking a grammar of the modelling language as input, JavaCC is a tool for automatic generation of Java sources implementing a lexer/parser for the specified language. JTB is a preprocessor to JavaCC. It also takes as input the grammar specification, but produces a JavaCC input file enhanced with Java code for supporting the Visitor Design Pattern [7], which is used for traversing the parse tree. The attribute repositories are implemented as MySQL database tables [11]. When a visitor function visits a node of the parse tree, it performs a look up action in a table, communicating with the MySQL database using a driver.

At the time in system development where the system model is built, the target platform is yet to be defined. Thus, the model must be represented independent of implementation details about concrete systems. Because possible platforms include hardware technologies such as FPGA, as well as distributed computing architectures, a modelling language able to model concurrency is required. As a modelling language, we are currently using the CSP inspired SystemC [8]. It is a superset of C++
extending the semantics with concurrent communicating sequential processes, ports, signals, and modules encapsulating processes. It is a suitable modelling language in a Component-Based Development framework due to its support for structured and hierarchic models, separation of functionality from communication, as well as the possibility for concurrency capture.

5 Example

For illustration of the principles of estimation and the application of our prototype tool, we consider an OFDM (Orthogonal Frequency Division Multiplex) receiver system as specified in various standards, e.g., the IEEE 802.11a wireless LAN standard [13]. Fig. 6 shows a simplified block diagram of the system. The baseband part of the receiver below the analogue-digital conversion is in focus. Initially, a serial-parallel conversion takes place in order to acquire a block of data for Fourier transformation. After the Fourier transformation, data is again serialized before demodulation by the appropriate scheme and Viterbi decoding. The numbers in Fig. 6 refers to the amount of data transfer between components.

The system belongs to the class of streaming data applications: it is potentially constantly employed. A model of such system must capture the potentially inherent parallelism of the application, even though the system may be implemented using a platform executing the individual components sequentially, for instance through multiprogramming supported by an operating system.

In this example, however, a Xilinx FPGA is chosen as the target platform. For the purpose of illustration of principle, a simple quality attribute, space consumption in terms of area is estimated. Other work, [3], has presented space consumption in terms of static code size for a 8051 micro-controller, hence motivating the choice of a different type of target platform here. The Xilinx platform is supported by a component library, Xilinx LogiCORE, which includes the components needed to assemble the receiver baseband system. This library serves as input to the database used by our prototype estimation tool. Table 1 shows area attributes for the individual components [12]. The area figures are the number of consumed configurable logic blocks (CLBs). For communication and synchronization between components, middleware components are added. We are currently evaluating middleware attributes for use in the estimation.
Table 1: Area attributes for the OFDM example.

<table>
<thead>
<tr>
<th>Component</th>
<th>Area [CLB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/P</td>
<td>32</td>
</tr>
<tr>
<td>FFT</td>
<td>1161</td>
</tr>
<tr>
<td>P/S</td>
<td>24</td>
</tr>
<tr>
<td>Viterbi</td>
<td>1084</td>
</tr>
</tbody>
</table>

Using the estimation tool on the SystemC system model, the visitor function is able to collect the area attributes for each functional component from the database. The analysis corresponds to summing up the attributes, and adding the overhead of the middleware component multiplied by the amount of communication since this information is present in the system model.

6 Conclusions

In this paper we have made a precise distinction between Component-Based Development and Model-Based Development. We have used this to develop a framework for estimation of qualities, or extra-functional properties, using a model-based approach. To perform estimation, the influence of the platform on the system qualities has been identified and incorporated into the estimation framework. These observations form the basis for an estimation tool, for which we have presented the structure and implementation details of a prototype.

The example deals with hardware components; but the approach carries over to software components as our terminology is technology independent. In addition, due to the model-based approach, an estimation tool as presented here applies to hardware/software codesign methodologies, as well as the development of systems-on-chip, which are typically constituted of both hardware and software components.

Estimation was discussed in the context of design space exploration. However, an estimation tool as presented here may serve other purposes in a development methodology using components. In many cases, some components need to be designed from scratch, as they are not available. A model-based estimation tool can assist in the specification of the required quality attribute budgets for the development of the missing components, so that the extra-functional properties of the composite system can be satisfied.

The tool has been initially tested with very simple analysis functionality: we have taken static modules, where simple addition suffices. This will hardly do for more dynamic properties, like dynamic memory consumption, power consumption
or quality-of-service in the time domain. Hence, future work needs to investigate composition functions for this kind of system qualities.

References


[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns, Addison-Wesley, 1995


Model-Based Deployment in Automotive Embedded Software: 
From a High-Level View to Low-Level Implementations

Andreas Bauer, Jan Romberg
Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
{baueran|romberg}@in.tum.de

Abstract
The electronic components in present-day automobiles are based on networks of electronic control units (ECU) running embedded software. The development of distributed, concurrent software applications based on such ECU networks is becoming increasingly complex and error-prone. In current practice, system-level views of the network are merely used to characterise technical constraints such as performance requirements, and to choose the hardware and software components accordingly. In contrast, the semantic integration of the distributed functions is typically deferred to later points in the development process, yielding a high effort for integrating and validating such distributed functions. To address in particular this issue, our paper advocates a more stringent use of high-level models based on distinct abstractions and a well-defined behavioural semantics. We introduce the corresponding notations and tools, and the overall methodology developed to support a stepwise development of distributed automotive applications. The paper then details on the issues of using such high-level models to facilitate deployment, and to obtain low-level implementations from integrated system models.

1 Introduction

Until recently, the electronic control system in a vehicle was mostly concerned with light switches, windshield wipers, or starter motors all of which were, more or less, realised as isolated systems provided from independent suppliers. Traditionally, the software for such embedded systems was implemented in a relatively low-level fashion as C, or Ada programs, and often directly in terms of native machine code. The last decade, however, saw an increasing use of integrated development toolkits such as ASCET [1], or the Simulink Real-Time Workshop [2] which facilitate reuse and provide retargetable generation of code based on dataflow models.
However, the nowadays increasing number of distributed ECUs in vehicles imposes fundamentally different problems for the automotive industry which is not tackled by the existing tool support alone. The sharing of data between ECUs that communicate via dedicated busses and bus protocols (e.g. CAN, MOST) allows the integration of additional functionality at lower costs. Consequently, this domain now requires different abstraction levels to be able to capture the actual dataflow between distributed ECUs inside vehicles as well.

Such higher-level models are also necessary to simulate and verify the behaviour and communication between ECUs to guarantee for safety and reliability of the deployed software. Ideally, the abstract models also facilitate reuse on various levels of abstraction. In a distributed system, isolated solutions at the level of programming languages are clearly not suitable for these requirements. However, high-level models raise a number of other issues though: for example, is it feasible to use them directly for code generation in a domain which, traditionally, confronts its users with limited computational resources?

Therefore in § 2, this paper first outlines such a typical target platform for (safety critical) embedded software as we encounter it prominently in the automotive domain. The actual partitioning and deployment issues are described in § 5 for which we first introduce abstractions and system descriptions that will also help classify the presented concepts in a realistic automotive industry context. Additionally, we briefly sketch the synchronously clocked computational model underlying our modelling formalism.

![Figure 1: A distributed target architecture.](image)

### 2 An Abstract Target Platform

Our abstract target consists of a network of ECUs connected via a bus. As can be seen in Fig. 1, each ECU is embedded into a host node which consists of the ECU itself, an operating system, a device driver module for interfacing the bus, one or many application tasks, and a dedicated communication layer.

The horizontal bar at the bottom of Fig. 1 indicates that the functionality contained within a high-level system model may be arbitrarily distributed among the
nodes of the network, i.e. distribution of a functionality is transparent in a top-down systems view. (Note that the terms “function” and “functionality” are used as synonyms in this context to describe a certain ability, or property of the system.)

The dedicated communication layer is merely a wrapper around the inter-task communication between applications of spatially separated ECUs. Its main purpose is to manage resources needed to buffer signals whenever necessary (see also §5.1). Communication itself, however, is handled by the device drivers which can be automatically generated for each ECU and protocol variant.

3 Abstraction & System Description

With the ongoing shift in the automotive industry towards distributed — and ideally reusable — software components, practitioners are not faced with a uniform system view anymore, e.g. source code. Components are now designed to be automatically deployed in a range of different vehicle types within a single class many of which offer, say, varying on-board electronic controllers as well as a different number of available ECUs for deployment. On a more abstract level, the behavioural view of the rather differently deployed components is expected to remain constant though. Fig. 2 illustrates how different abstract views on automotive software components can be assimilated to a common integrated system model.

![Figure 2: Abstract system views.](image)

The view on functional dependencies is, typically, the most abstract model of an automotive software system. It captures the structure as well as the functional dependencies common to a class of vehicles by the same manufacturer. A component view, on the other hand, contains the internal interaction patterns of individual software components in terms of dataflow, communication and behaviour. This is already sufficiently expressive and detailed to allow for validation and simulation of designs, while an operational view, typically, contains aspects which are unique to the actual target platform. Naturally, reuse of components gets increasingly difficult with a decreasing level of abstraction.

Each level needs to be associated with a number of custom description techniques, first to allow for independent top-down systems design, e.g. abstract defini-
tion of sensor and actuator components, and secondly for a subsequent refinement down to a mapping onto actual hardware.

**Functional Dependencies:** Common to this view are structure oriented views, i.e. system structure diagrams (SSD), to describe the overall structure of a system. Typically, SSDs are specified as hierarchical component networks where components communicate via typed and directed channels and typed ports, similar to the visual representation of UML-RT [3] and some Architecture Description Languages.

**Component View:** In this abstraction, we require a description of the individual software components to be complete with respect to behaviour. Therefore, the employed description techniques, typically, include state transition diagrams (STD), low-level dataflow diagrams (DFD), or more message-oriented diagrams (see also [4]). DFDs can be viewed upon as a refinement of SSDs and describe the algorithmic dataflow occurring during a computation. They consist of components performing the computation (i.e. blocks), interface elements of those components (i.e. ports), and connections between those interface elements (i.e. channels).

**Operational Model:** The operational model employs a similar visual representation as the component view — cluster communication diagrams (CCD) — but it is an implementation-driven refinement containing those details essential for deployment. CCDs then represent the main computational units (i.e. clusters, abstract tasks) that interact directly with the real-time operating system (scheduler) and the dedicated computational layer; that is, clusters are the least distributable units from the integrated system model: clusters are not split across two tasks and are always connected using explicit signal sampling operators (see § 4 and § 5.2). In this context, however, clusters must not be confused with TTP-clusters [5].

Note, at this point, we abstain from giving a more detailed description of the individual visual representations and their exact properties as the important graphical notations relevant for this paper are introduced in § 4 and § 5 by practical examples, respectively.

### 4 Computational Model & System Behaviour

The behavioural model of the systems described in this paper is that of current AUTOFOCUS [6, 4] models. It is based on the synchrony hypothesis using a discrete notion of time. The synchronous paradigm [7, 8] basically states that a system reacts to external stimuli within one instant, i.e. the delay between internal computations cannot be observed. This approach has enjoyed widespread acceptance in the control
and hardware design domains, and is largely compatible with the commercially established tools such as ASCET, or Simulink. As opposed to several other approaches used for real-time specification and programming, the discrete-time semantics and deterministic concurrency keep behavioural evaluation of large designs manageable. The AutoFocus framework is based on such a deterministic time-synchronous interpretation: components communicate through timed streams, where each stream uses the same global time base.

In order to support the multiform event patterns and frequencies observed in typical real-time systems, each stream of signals is associated with a clock. Similar to other synchronous dataflow languages [7], an AutoFocus clock can be thought of as a boolean stream that merely indicates whether a value is currently present (tt), or not (ff). Clocks characterise streams both external, such as frequencies imposed by surrounding actors or real-time constraints, and internal to the system: by using clock inference properties the internal clocks can be inferred from the according inputs, respectively. (Think of the integration of black-box “legacy components”, for example.)

Our current tool prototypes provide both automated inference of internal clocks and static checking of well-formedness of the model, i.e. detecting absence of causal cycles and a soundness verification of clocks. The implementation is very similar to that of a static type system in strongly typed programming languages.

In AutoFocus, each clock is defined w.r.t. a base clock, k, which is the fastest clock in and underlying a system; that is, the most fine-grained time scale upon which a system reacts to external stimuli. The base clock itself is represented by the boolean expression tt, i.e. the expression that evaluates to true at any instance of k. A model’s clock expressions are typically ordered using a ≤-relation.

![Figure 3: Explicit signal sampling in DFDs.](image)

Furthermore, in AutoFocus it is not only possible to infer clocks, but also to make up new ones based on other clock expressions. The DFD given in Fig. 3 bears an explicit when operator which samples the input stream a to the rate of boolean stream b; that is, a' = a whenever b evaluates to tt. The output and input ports are depicted by black and white rectangles, respectively.
In accordance with the notion of using clock expressions, all of a system model’s entities can be represented using a dedicated language based on expressions. Consequently, expressions in AUTOFOCUS range over channels, ports, and combinations thereof using dedicated operators.

Let $Exp$ be the set of all such expressions used in a system model and let $Exp_B$ denote the set of all boolean expressions. We can now introduce a function $ck$ which gives us the actual clock of any $e \in Exp$:

$$ ck : Exp \rightarrow Exp_B. $$

**Example.** To illustrate how clocks are put into practice, let’s assume that the following virtual values are being transmitted in the model as it is given in Fig. 3 where $\tau$ denotes an absent signal/value:

$$
\begin{align*}
  a &: 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ ...
  b &: \texttt{tt} \ \texttt{ff} \ \texttt{tt} \ \texttt{ff} \ \texttt{tt} \ \texttt{ff} \ \texttt{tt} \ \texttt{ff} \ ...
  a' = a \ \text{when} \ b &: 1 \ \tau \ 3 \ \tau \ 5 \ \tau \ 7 \ \tau \ ...
  c &: 0 \ \tau \ 1 \ \tau \ 2 \ \tau \ 4 \ \tau \ ...
\end{align*}
$$

Obviously, $a$ never yields $\tau$, indicating that its clock is exactly the system’s base clock. Here, sampling is necessary since $F$ expects its inputs both at the same pace as $c$: every second “tick” relative to the base clock. Hence, the $\text{when}$ operator projects the stream $a$ to the slower clock explicitly defined by boolean stream $b$. Note, however, up sampling works accordingly and is achieved using the same operator. □

In order to allow for well-defined feedback loops and to provide memory slots holding temporary values, explicit delay operators are necessary. Fig. 4 depicts a model which makes use of an explicit delay block (black and white diamond shape) that behaves as follows: a value is held for one clock period respectively; the period is determined by the clock speed of the stream setting that value.

![Figure 4: DFD with delayed signal.](image)

Here, the delay is used to “feed back” a previous value of $F$’s computation, $b$. Each delay block is associated with an initial value. Note that the clock of the delayed signal equals the clock of the original signal. That is, the clock of $a'$ is that of $b$, and if $ck(a) = ck(b)$, then $ck(a) = ck(b) = ck(a')$.
5 Partitioning and Deployment

One inherent property of SSDs in AUTOFOCUS is the underlying assumption that communication between components is always delayed (i.e., each connecting channel contains exactly one implicit delay operator). This property of SSDs enforces the introduction of “predefined breaking points”, which will be needed on the Operational View level to partition the design into individual tasks. From a methodological point of view, this definition facilitates the individual and also more independent development of each specified component. In the graphical notation, delayed communication is expressed with rounded ports (see Fig. 5).

![Figure 5: Example SSD.](image)

We will show in the following that the introduction of explicit delays in early stages of development in our time-synchronous system model are prerequisites for deploying a distributed application across several tasks, or even across network of ECUs.

5.1 Communication Layer

Instead of dealing directly with inter-task communication, data-consistency, and I/O handling, we define an abstract communication layer that “wraps” all read and write accesses, respectively (see §2). This layer acts as a kind of middleware providing basic communication services and data integrity to the application’s tasks running on it. In particular, the middleware provides a write handler (similar to SendMessage() service in OSEK COM [9]), and a read handler for messages (similar to ReceiveMessage() in OSEK COM).

Effectively, the layer constitutes a transparent communication model for each node and the tasks running on it, i.e., its technical realisation ensures that sufficient heap (register, or buffer) space is allocated when messages need buffering as is the case, for instance, when tasks with different clock speeds exchange signal frames. The following prerequisites are essential for the communication layer to yield the desired behaviour in practice:

- Execution of an accurate static analysis for minimal message allocation,
and predetermination of an appropriate task scheduling algorithm.

Our AutoFocus-based prototypes already provide for the former by allowing the static analysis and by associating appropriate memory with each delay operator in the model. The exact required amount of temporary space in total is then determined by a subsequent “clock comparison” of the communicating tasks (see §5.2).

For the remainder of the paper, we assume a rate-monotonic scheduling policy, based on an operating system with a fixed-priority preemptive scheduler, where task priorities may be statically assigned; the latter limitation is, for instance, imposed by the OSEK standard [10] for automotive operating systems. Rate monotonicity simply asserts that tasks with smaller periods are assigned higher priorities than tasks with greater periods [11].

5.2 Variables and Message Slots

The operational system abstraction/view, as sketched in §3, contains the transition from the hierarchic and connected SSD components to a clustered system view yielding all delay and sampling operators; that is, relevant implementation details.

The CCDs then present a flat description of the time-synchronous system model which allows for the static analysis of the heap (register, or buffer) consumption in terms of message buffers as well as for (almost) arbitrary partitioning variants: unlike SSDs which are grouped according to conceptual coherency and as reusable units, CCDs are typically partitioned to either

- yield a maximum of technical efficiency in the implementation,
- to account for physical proximity of an application part to sensors and actuators, or
- to adapt the software structure to other non-functional requirements, such as fault tolerance requirements.

There are also cases where it is required to partition CCD clusters along the same boundaries as SSD components. For instance, if components A and B are known to be always mapped to different processors, then the clusters A and B should be fully disjoint, i.e. there exists no cluster containing parts of both A’s and B’s functionalities.

Note that the semantics of SSD composition, i.e. every channel incorporates a delay, ensures that the following delay constraints for CCDs are met given rate-monotonic scheduling and when communication follows the boundaries of the SSD components. Let A and B be a sending and a receiving cluster, respectively:

\[ ck(A) = ck(B), \]

i.e. equally fast clocks. In this case, the priority of A’s and B’s tasks are the same, so communication occurs delayed; in effect, the communication layer needs to provide two message copies to avoid data inconsistencies.
$ck(A) < ck(B)$. When the clock of $A$ is faster than the clock of $B$, i.e. the period is smaller, we may use undelayed communication; only one message copy is needed.

$ck(A) > ck(B)$. Communication is delayed, when the clock of $A$ is slower than the clock of $B$; in this case, two message copies are needed.

Obviously, the above comparison of task periods and static memory analysis is only possible by extending the clock associations from individual ports, or channels to the entire clusters themselves. Therefore, for periodic designs, a cluster clock is inferred as the “greatest common divisor” (gcd) of its individual clock periods. Note that internal clocks cannot — by definition of blocks and DFDs — be faster than the fastest external clock, so considering the clocks of incoming channels/ports in order to determine a cluster clock is fully sufficient. The following examples elaborate on that.

![Diagram](a) Undelayed CCD.

![Diagram](b) Copped need-provide interference-polygon (grey).

Figure 6: Fast cluster writes to slower cluster.

**Example (fast cluster → slow cluster).** In Fig. 6(a) an example CCD consisting of two clusters $A$ and $B$ is depicted. For the sake of simplicity, we only consider
periodic clocks, and write the clock periods next to the corresponding ports. By attaching a label $x:6$, we indicate that a port $x$ holds a value every 6th tick relative to the base clock; that is, in our case we obtain $ck(A) = gcd(6, 3, 3) = 3$. $A$ writes signal $a$, which is sampled by a when-operator and read as signal $b$ by cluster $B$. Communication between $A$ and $B$ is not delayed.

Furthermore, let’s assume that cluster $A$ corresponds to a task $T_A$ with a period and deadline of 30ms, and that cluster $B$ corresponds to a task $T_B$ with a period and deadline of 60ms. In other words, $T_A$ and $T_B$ are each released periodically at the beginning of their respective cycles which are indicated in Fig. 6(b) by black horizontal lines.

Both tasks are executed on the same ECU, and are scheduled according to the rate monotonic policy, i.e. $T_A$ has a higher priority than $T_B$. In order to avoid data inconsistencies, for any step $k$, $T_B$ needs a stable value $b_k$ during the whole duration of its period. In the time scale, this is indicated by the grey “need $b$” bar.

A new value for port $a$ is provided periodically by $T_A$, indicated by the “provide $a$” bar. Note, for any step $k$ of $B$, the “provide $a_k$” bar starts chronologically after the “need $b_k$” bar.

Because of $T_A$’s higher priority, $b_k$ will never actually be read before $T_A$ has finished its computation, and $a_k$ has been written. We indicate this by a dashed bar for “need $b$” during $T_A$’s activation. Therefore, we can safely associate $a_k$ with $b_k$, which corresponds to immediate communication in the model. Since the written variable and the read variable correspond to the same memory location, the communication layer does not have to perform an explicit message copy operation. This example has shown that communication from fast to slow clusters does not require the introduction of additional delays in the model.

\[\]

![Figure 7: Slow cluster writes to faster cluster.](image-url)
Example (slow cluster → fast cluster). Fig. 7(a) depicts a CCD with two clusters $A$ and $B$. The overall cluster clock of $A$ is 6, $B$’s is 2. Now the slower cluster $A$ writes to the faster cluster, and the clocks are in a relationship $ck(A) > ck(B)$. According to the rules on page 100, an explicit delay is imposed in such cases, indicated by the diamond-shaped operator between $a$ and $b$.

Fig. 7(b) shows how the delay relates to the time scale of two associated tasks $T_A$ (period/deadline 60ms) and $T_B$ (period/deadline 20ms). Since $ck(A) > ck(B)$, the “need b” period can be safely extended to $T_A$’s period of 60ms.

This illustrates that if all tasks meet their respective deadlines, for any step $k$, $T_B$ will never read $b_k$ before $a_{k-1}$ has been written. We can, therefore, safely associate $a_{k-1}$ with $b_k$ for any $k$, corresponding to a delay in the model. The black double-headed arrows indicate explicit message copy operations performed by the communication layer.

6 Conclusions & Summary

In this paper we have shown that deployment related issues in the development of distributed automotive controlling software, like insertion of explicit delays in a time-synchronous system model, must not necessarily be driven in a bottom-up manner, but can also be asserted high-level and from a top-down perspective. Given the underlying assumptions regarding schedulability and the various static analyses, the introduction of delay operators in early development stages through the use of SSDs yields several advantages: firstly, the delays constitute predetermined breaking points in subsequent refinement and implementation processes, and secondly upon partitioning and clustering of the components, delays must not be added manually, i.e. the original communication structure remains mostly unchanged. The latter is particularly important, because essentially it means that a formerly verified behavioural model of the system, remains stable in the final implementation; all the implementation’s delays have been present in the structural view as well. This lowers the validation and verification efforts drastically and increases the reusability of components.

Although, as we have sketched in §5.2, delays are not always essential to support, say, the writing of a fast cluster to a slower cluster. However, early assertion of a delay does not alter the communication’s behaviour if inserted after the down sampling operator that lies in between the CCD clusters. What is more, in that case it is theoretically possible to assert an arbitrary amount of delay operators after the down sampling occurs; the result being a higher memory consumption due to excessive message buffering.

On the other hand, this example illustrates that top-down asserted delay operators do not necessarily guarantee for the most efficient implementation of a distributed application. In fact, this paper comprises a trade-off between these very aspects of optimisation and the advantages of having separate, reusable and verifiable system components. In other words, using the presented methodology results in lower veri-
fication efforts on the one hand, and in a less efficient implementation on the other.

Furthermore, in §3 we have introduced and sketched several graphical notations to support the presented development process of distributed embedded systems: a hierarchical SSD description to capture a system’s overall structure, DFDs to express a component’s computation and dataflow, and CCDs to explicitly visualise deployment details and to facilitate partitioning according to, say, “clock boundaries”, or SSD component boundaries. (Compared to SSD-driven partitioning, a clock-driven strategy groups clusters according to common clock speeds which often results in faster implementations.)

Editors for the discussed notations, the key algorithms underlying the analysis (e. g. clock inference and well-formedness checks) and the various abstract system views are already supported by a tool prototype based on the existing AUTOFOCUS framework.

Acknowledgements

We would like to thank Timothy Bourke, National ICT Australia, Sydney, who read and gave valuable feedback on early drafts of this paper.

References


A Model-Based Approach to the Development of Distributed Control Systems

Manuel Bernardo Barbosa (mbb@di.uminho.pt)
João Miguel Fernandes (jmf@di.uminho.pt)
Departamento de Informática, Escola de Engenharia
Universidade do Minho, Braga, Portugal

Abstract

Distributed Control Systems (DCS) are a class of application with specific characteristics. This type of system is used in industrial environments to control manufacturing processes. Usually they comprise a controller, a fieldbus network, and a set of Of-The-Shelf (OTS) components, interfacing process signals with real-time QoS requirements. In this paper we present a Model Driven Development (MDD) method that targets this category of systems. This method focuses on the critical stages of DCS development. Namely, the specification of system requirements, the choice of OTS modules and fieldbus system, and the validation of the design using real-time analysis tools. This MDD method uses the Unified modelling Language (UML) as support notation, including the extensions defined in the UML Profile for Schedulability, Performance and Time Specification.

1 Introduction

The use of MDD methods greatly improves the productivity and reliability of software systems. There is no reason why this type of methodology should not bring similar benefits to the development of software for embedded systems. Previous experiences show that this is indeed the case.

These issues are particularly relevant in the development of embedded systems with real-time QoS requirements. In this paper we present a MDD method that targets DCS, a type of real-time system common in industrial environments.

A DCS is composed of several intelligent modules connected to sensors and actuators that provide an interface to the process being controlled. These modules cooperate by communicating through a network system, typically a fieldbus. They may

*Work partially supported by Fundação para a Ciência e a Tecnologia (FCT) and Fundo Europeu de Desenvolvimento Regional (FEDER) under project “METHODES: Methodologies and Tools for Developing Complex Real-Time Embedded Systems” (POSI/37334/CHS/2001).
be custom-built for a particular application but, in most cases, standard OTS components are used. The overall operation of the system is managed by one or more controller devices, which may also operate as gateways to other information systems.

The need for this type of methodology became apparent in our interaction with an industrial partner that develops DCS. Traditional software development MDD methods and real-time system development methods are not able to solve all the problems raised by the peculiarities of this type of system.

Our MDD method addresses specific aspects of the development of DCS: the identification of process output/input signals, i.e., system input/output signals; the selection of the OTS components and fieldbus system; the assignment of input/output signals to OTS components; the association of input/output signal events with network message transfers; and the usage of real-time analysis tools.

It uses the Unified Modelling Language (UML) and it is based on the UML Profile for Schedulability, Performance and Time Specification [7].

2 Background

2.1 Real-Time System Modelling Using UML

The central difference between modelling a software application and modelling a real-time system is that it is not possible to model the latter unless quantitative data can be depicted in the model. In fact, for most software applications, it suffices to model functional requirements. For real-time systems, it is also necessary to model real-time QoS characteristics consisting of time-related quantitative information. Even though UML is particularly well suited to develop real-time software [9], a careful adaptation of UML is required due to the particular characteristics of this type of software.

The usage of UML in the embedded field dates from 1998 [1]. In his book, Douglass introduces techniques for developing embedded real-time systems with UML. Another important line of research is the UML-RT proposed in [8] [11]. UML-RT is strongy based on ROOM [10] and proposes an approach based on collaboration diagrams. The adoption of UML-RT to model embedded real-time systems in the telecommunication domain is discussed in [5]. Other experiences, such as that in [4], show that UML is suitable to provide notational support in tool-chains for real-time embedded system design.

Recently, another step forward has been taken. OMG defined the UML Profile for Schedulability, Performance, and Time Specification [7]. This profile was written to be applicable to a wide range of application areas. It does not attempt to define a complete set of real-time modelling concepts. This would be virtually impossible given the variety of systems that fit this category, the many design styles that are used and the different modelling approaches that can be used. For example, only the designer can determine what is an acceptable level of detail when modelling a
particular system. The concepts that are defined in this profile are kept to a minimum and at a level of abstraction that allows designers to take advantage of the “power of UML”.

This profile also takes into consideration a very important part of real-time system development: automatic system analysis. System analysis consists of producing a specialized view of the system that can be processed automatically for one of two purposes: calculating the values of parameters still missing in the model; or assessing a particular characteristic of the system depicted in the model.

2.2 Distributed Control Systems (DCS)

This work focuses on the development of DCS, a type of process control system that is common in the manufacturing industry. These systems are used to control manufacturing processes by collecting process data and acting on the process according to a specific control algorithm. Figure 1 shows the typical structure of this type of system.

Systems like this present a set of common characteristics that permit tailoring a development process to them:

**Self-containment** Restrictions imposed by the control functionality usually imply that the control loop operation must be isolated from external interaction. In most cases, the whole system will present to the exterior a very simplified interface to allow for the collection of operational data, and the setting of global operational parameters.

**Real-time QoS requirements** Invariably, the specifications for this type of system include real-time QoS requirements. These will apply to sensing and acting on process signals and, consequently, to all the processing that must be carried out to acquire, process, generate and transfer the associated data.

**Fieldbus communication system** The geographical locations where data collection and actuation must be performed are spread around the manufacturing
floor, separated by distances ranging from a few to several hundreds of meters. Fieldbus standards such as Profibus and CANopen have been developed specifically to address the needs of this type of control systems [2].

**OTS embedded components** The economic benefits of using OTS components compatible with a particular fieldbus standard are well known. They include reduced system development time and cost, minimal downtimes through the replacement of damaged or faulty components, etc. However, the use of OTS components introduces a new problem in real-time system design: the degree of confidence that may be placed on the overall system greatly depends on the confidence placed on the reliability of the components and on the degree of knowledge that exists regarding their internal operation.

**Fault-tolerance QoS requirements** Some kind of fault tolerance guarantee is usually implied in the system specification and, often, it will be stated explicitly. Our project has not yet reached the point where fault-tolerance issues are addressed and this work does not cover fault tolerance requirements.

### 3 Development Process Overview

The following technological or architectural levels can be distinguished in the development of a DCS:

1. Hardware/software partition and hardware design
2. Real-time operating systems
3. Real-time network and OTS device software
4. Controller device software
5. Application software

Clearly, from the system characterisation in Section 2.2, the technological solutions for the first three levels will be very much constrained by the system specification itself. This in fact means that the entire architecture of the system will be a rather direct consequence of the system specification. At these levels, the designer will be left with choosing and customizing suitable commercial products: OTS components and fieldbus system.

Software development will, in most cases, be concentrated on the the upper levels. It will comprise solely the software running in controller devices and connecting the control system to the outside world e.g. user interfaces, database connections, gateway connections, etc.

A development process for this type of system does not have to be built from scratch. In fact we chose to base our work on the embedded system development
process presented in [3]. However, it was necessary to customize this process to address the peculiarities of DCS development. We have defined six development stages that will be described in the following sections.

3.1 Black-box system requirements

A fundamental aspect of system analysis is a correct characterisation of the requirements for the system’s interaction with the environment.

In the case of DCS three types of external entities may interact with the system:

- **Human users** – They observe and adjust system operational parameters. In this case, typical user interaction requirements, common to other areas of software development, are at play.

- **Input and output process signals (or logical groups of signals e.g. machines)** – This is the process interface itself. The system requirements include the real-time characteristics of the signals that the system must acquire from sensors and generate for actuators.

- **Other systems** – These may collect monitoring information and adjust operational parameters. In this case requirements may be of two types. If the peer system is a higher-level system, then the application is functioning as a gateway, and the requirements are as for human users. If the peer system is at the same level as the one being designed, it is possible that real-time requirements apply to this interaction as well.

For each input or output process signal, a minimum set of parameters must be identified: the physical nature of the signal, the timing characteristics of the interaction, and the operations that must be perform on that signal.

3.2 OTS device and fieldbus selection

One immediate consequence of using OTS components is that the selection of the commercial components and fieldbus system that will be used in the system becomes a fundamental stage in the design process.

The selection of this type of equipment for a particular application may be subject to constraints that are out of the scope of a software development process e.g. economic and strategic issues. It is clear that the choices made by the designer, at this level, also depend on restrictions imposed by the physical environment on which the system will operate. An extensive discussion of the aspects that must be considered in this design stage can be found in [2].

More relevant for this discussion is an assessment of how the parameters resulting from the system analysis described in the previous section affect this design stage. The identification of the required OTS components implies that these components are assigned to a set of input and/or output signals which they will be interfacing.
The characteristics of these signals will be determinant in the selection process. The nature of the signal will of course require that a compatible module is selected e.g. if we want to sense an analog signal, we must use an analog input module. The real-time QoS requirements for sensing or actuating on the signal will function as minimum performance requirements for the OTS module. Whether or not the device will be capable of doing that in run-time can only be assessed by analysing the entire system, as will be discussed in Section 3.6.

3.3 OTS device configuration and real-time model

OTS components are themselves complex subsystems, usually designed by a third party. These components present some degree of configurability, which the system integrator must customize to ensure that the overall system will operate correctly.

The configuration of each component will be performed in an implementation-specific manner. However, the designer will have to derive the configuration data for all of the components from the requirements associated with the signals that each of them will be interfacing. This is done using an algorithm that treats all devices uniformly. Usually this type of algorithm is specific to a particular fieldbus standard. For example, for a particular type of network, the messages that a device will be transmitting or receiving are configuration parameters that apply to all devices.

We believe that, to a great degree, the derivation of the configuration data for OTS devices can be automated. Nevertheless, for this to be possible, all devices compatible with a particular fieldbus will have to be modeled consistently, at a suitably high level of abstraction, based on coherent configuration parameters.

Furthermore, a very important phase in the design of DCS is the real-time analysis of the entire system. This type of analysis requires all of the system’s components to be described in sufficient detail, in order to determine their influence in the system’s behaviour. A more detailed model of the system will allow for a more precise analysis and, consequently, for less pessimistic design choices.

For the reasons explained in the previous paragraphs, our development method requires a generic model for the OTS components that are used in the DCS. In this paper we will be focusing on components that use the CAN network [6] as a communication link. This discussion will apply to any higher-layer protocol [2] using CAN, and it can also be easily adapted to other fieldbus technologies.

The generic model we use is shown in Figure 2. It reflects a view of a CAN-based OTS device where its internal behavior is structured according to the CAN messages the device is configured to receive and transmit.

This model is unlikely to be a truthful description of the internal structure of every component. However, it provides just enough detail to permit analysing the influence that each component will have on the overall operation of the DCS. The simplicity of this model also reduces to a minimum the number of performance parameters that must be known about a given implementation. Ideally, these parameters should be
Figure 2: Model of an OTS CAN device
3.4 Network configuration and real-time model

One of the determinant factors in the development of DCS is the influence of the communication infrastructure (fieldbus system) in the operation of the system.

Fieldbusses are communication protocols which operate over a shared medium, typically in bus configuration. A fieldbus influences the operation of the system due to the serialisation that it imposes on message exchanges: typically, only one device can send one message at a time. This implies that other devices wishing to transmit at the same time will be subject to delays in their operation. Conflicts are usually solved using a priority system.

This type of operation can be modeled as a resource sharing process that is common in concurrent systems. Conceptually, the network can be seen as a shared resource, residing on an independent node. When a device wishes to transmit a message, it must access the shared resource. The resource will be blocked until the message is transmitted. Bus access conflicts can be emulated through the priority system that is usually associated with this type of resource sharing mechanism.

The class diagram in Figure 3 shows the class that is used to represent the CAN bus, and its association with multiple CAN controller objects.

For each CAN bus that is used in the DCS (usually there is only one), an instance of the MCANbus class will be present in the model. Each CAN interface connected to the network (usually one per device) will be represented by an instance of one of the CAN controller classes shown in Figure 2. The network activity in the system will be represented in the model through message exchanges between these object instances, as will be seen in Section 4.

3.5 Controller development and real-time model

The application running on the controller(s) coordinating the operation of the DCS will be the only parts of the software composing the system that will actually be designed by the system developer.

Typically, a controller in this type of environment is itself a complex embedded system. The description of a complete development process for this type of module
is not the purpose of this paper. Suitable MDD methods exist [3], and can be used without change.

One additional point that must be considered is that the controller’s influence on the global behavior of the system must also be taken into account. This means that, similarly to what was described in Section 3.3 for OTS components, a suitable model of the controller must be produced.

Typically, the resulting model will not be too different from the class structure shown in Figure 2. Controllers produce and consume messages like all the other nodes. The difference being that the data contained in these messages is usually generated and consumed by the controller itself. There will however be additional functionality associated with the controller’s interaction with the outside world, which must also be included in the model, in order to accurately characterise the workload carried out by the corresponding node.

3.6 Global real-time analysis

Global real-time QoS requirements appear as end-to-end time constraints, that apply to complex sequences of interactions, between multiple object instances in the complete system model. These end-to-end time constraints put limits on the time that may elapse from the time instant at which a triggering event occurs, until the system’s response to that event is completed. These limits may represent hard or soft deadlines for the system’s response.

Real-time analysis is a fundamental stage in the design of a DCS with real-time QoS requirements, where we validate that the solution meets its requirements. In some cases this analysis may even include the generation of operational parameters that can only be calculated based on a global view of the problem e.g. the assignment of priorities to messages circulating on the network to guarantee response times.

Our model-driven approach must support this type of analysis. This means that the models must include all the information that is required to carry out the necessary validation. Moreover, it should be possible to show, within the model, relevant results obtained during the real-time analysis.

The real-time analysis is usually carried out by specialised tools, which present an application-specific interface. Nevertheless, it is reasonable to assume that, as long as such tools are able to import and export analysis information to data files, it is possible to implement a design environment such as the one shown in Figure 4. An example of such platform integration can be found in [4].

A real-time analysis tool for CAN-based systems is being developed within the Methodes project. It will implement a data import/export feature that will allow it to read/write real-time analysis data from/to XML files. This data will be extracted from and reinserted into XMI representations of UML models by an additional tool, currently being developed at Universidade do Minho.
4 modelling Approach

One trait that will be common to all stages of the design process is the need for a language to document system requirements and design solutions at all technological levels. For the reasons presented in Section 2, UML is a natural choice as the notational support for this MDD method. The following sections describe our use of UML throughout the development stages presented in Section 3.

The diagrams that will be presented constitute a small example that intends to demonstrate our development process. Due to limitations in the UML editing tool that was used to produce these diagrams, stereotypes are shown within note elements, together with the tagged values to which they are related. For the same reason, whenever note elements are associated with messages, this association is indicated by including the name of the message in the appropriate note element.

4.1 Black-box system requirements

At this level, the modelling is done through Use-Case Diagrams, Sequence Diagrams and Activity Diagrams. These diagrams are used to depict the intended interaction of the system with external entities or actors.

4.1.1 Use Case Diagrams

Use Case diagrams can be used without any adaptations to depict the system’s interaction with the exterior. Each external entity appears as an actor, interacting with the system in a way that is put into context by the use case’s name.

For use cases where no real-time QoS requirements apply, usually those relating to user interaction or gateway operation, the modelling process is the classical one used in the UML. Sequence diagrams can also be used in the usual way to further detail the functionality associated with the use case.

For process interactions, two stereotypes called MInputSignal and MOutputSignal were defined. They modify actors to represent input and output signals (from the
Figure 5: Use-case diagram

system’s point of view). Figure 5 shows a use case diagram where a system interfaces four process signals. The actors representing the signals are modified with a stereotype that specifies the nature of the signal. Associated with the stereotypes that identify the process signals are the \texttt{MsignalType} and \texttt{MsignalBits} tags, which are self-explanatory.

This type of information can also be represented using a signal table that includes all relevant signal characteristics. In this case, if the timing characteristics for the signal are also included in the table, sequence diagrams may be omitted for these use cases. This approach would be similar to the one described in [1].

4.1.2 Sequence Diagrams

Sequence Diagrams are also used without major changes to provide detailed descriptions of the system functionality represented by a particular use case. However, for use cases with real-time QoS requirements, additional information must be included. This information consists of performance QoS requirements and, therefore, we chose to use the notational extensions defined in the performance analysis part of [7]. Each sequence diagram depicts an instance of an actor/system interaction. Represented in the diagram are the system itself, plus all actors representing related signals.

Input (resp. output) signals are represented as a message from (resp. to) the actor to (resp. from) the system. These messages are called \texttt{update} and they represent signal value events that must be read by (resp. produced by) the system. They are stereotyped as \texttt{PAopenLoad} and an occurrence pattern and required response time are also indicated. Figure 6 shows two examples.

If there is a cause/effect relationship between two signals, or if their interaction with the system is somehow related, they should be shown in the same sequence...
4.1.3 Activity Diagrams

For each signal acquired or applied to the process, it is necessary to specify what the control system will do with the input signal values it consumes, and how these are used to produce new output signal values i.e. the control algorithm. This is best achieved using activity diagrams.

The examples in Figure 7 show how activity diagrams can be tagged using the stereotype described in the previous section, to show required end-to-end execution times.

4.2 Representing DCS architecture

The internal architecture of the system, showing the network infrastructure and the selected OTS devices can be represented using Deployment Diagrams. In our approach, each device is represented as a node. Stereotype MOTSComponent has been defined to identify OTS components, as shown in Figure 8.

Note that the process signals identified in the requirements analysis are also
shown in this diagram as actors associated with one of the OTS components. This is how signal assignments are represented. Also note that the communication infrastructure is represented by a special node stereotyped as \textit{MCANetwork}. Other nodes shown in the diagram in Figure 8, and not stereotyped, will be hosting controller software developed during the design process.

The (modeled) internal structure of each OTS component (introduced in Section 3.3) is described through additional Deployment Diagrams such as the one shown in Figure 9. This diagram shows the objects that emulate the OTS device internal software. In this case, the OTS component interfaces one input and one output signals, receiving one message, and transmitting another one.

4.3 Real-time modelling

The real-time QoS requirements and characteristics of a system depend on run-time factors, the most relevant of which are the processing platforms hosting the running
software and the network activity on the fieldbus system. Additionally, the real-time characteristics of a system apply to interactions between object instances. These properties are not inherent to the classes involved, but to each instance itself [3]. In this paper we will be focusing on the interactions that are relevant for the real-time characteristics of the system, although many more interactions will exist in the model.

A real-time analysis consists of a schedulability analysis over a specialised view of the developed system. This is why the tags and stereotypes that we use to represent the real-time analysis data are taken from the schedulability part of [7]. The diagrams presented in this section were tailored for a specific real-time analysis technique. A small description of the parameters required by this technique are included in Section 4.4. Nevertheless, these models are sufficiently generic to be easily adaptable to other real-time analysis tools and even generic schedulability analysis tools that are compatible with [7].

Two UML diagrams can be used to depict interactions between object instances: Sequence and Collaboration Diagrams. Although they are interchangeable, here we will be using only the latter. Also, in collaboration diagrams it is possible to position object instances so as to emphasise deployment associations.

The collaboration diagram in Figure 10 shows the real-time parameters associated with one of the CAN messages in our example. Note that message exchanges are stereotyped differently.

One collaboration diagram similar to this one is included in the model for each CAN message transferred on the bus. Each of these diagrams depicts the complete sequence of object interactions associated with the corresponding CAN message. Two types of messages can be used as triggers:

- Process signal events that give rise to CAN message transmission (either event-driven or timer-driven).
- Internal device events that, for some application-specific reason, trigger the
transmission of CAN messages on the bus.

The workload associated with the scheduling job is composed of several message exchanges stereotyped as \textit{SAAction}. There are two different uses for this stereotype:

- Software execution, as in the case of the \texttt{main} methods, which represents computational load.
- CAN message transmission, as in the case of the \texttt{distribute} function, where the load is associated with bus access and transmission delays.

The objects stereotyped as schedulable resources correspond to independent threads of execution inside a particular device. Collecting the different threads that are identified in the collaboration diagrams created for each CAN message, and correlating this information with the architecture information presented in Section 4.2, an analysis tool is able to enumerate all the threads operating concurrently in each device, their relative priorities and timing characteristics.

Finally, the objects stereotyped as shared resources in this diagram are all related with the CAN communication infrastructure. The object instance representing the CAN bus appears in all collaboration objects involving CAN communication in that particular bus. This resource is shared by all CAN controller object instances to which it is connected. The combination of the bus access (the \texttt{distribute} method) operations throughout these collaboration diagrams allows an analysis tool to work out bus access conflicts, task blocking due to collisions, and bus utilisation parameters. The object instances representing CAN controllers are shared, within the host device, by message transmission and message reception routines.

The object instance representing the CAN bus is conceptually deployed on a node stereotyped as \textit{MCANnetwork}, as shown in the diagram in Figure 11. The \textit{MCANNetwork} stereotype is derived from the \textit{SAEngine} stereotype defined in [7]. It is seen as a computational resource with a particular processing speed (in this case the bus rate), schedulability and utilization. Messages transmitted on the bus are modeled as processes executing on this conceptual processor.

### 4.4 Interfacing real-time analysis tools

The collaboration diagrams presented in this paper constitute a small example of a DCS. This example was created for the real-time analysis tool being developed
In our implementation, this information is extracted from an XMI coding of the UML model, into an XML file which can be imported into the analysis tool. This extraction is performed by combining the structural information provided by the deployment and collaboration diagrams presented in Section 4 with the quantitative data specified using the stereotypes and tagged values also there described.

Note that the analysis results can be put back into the model using the reverse procedure. For example, in Figures 10 and 11 there are several tags that do not present numerical values, but variable names beginning with $. Analysis results can be traced back to these variable names and presented as the values of the corresponding tags.

5 Conclusions

DCS are a class of application with specific characteristics, such as the use of OTS components and fieldbuses, and the existence of real-time QoS requirements. In this paper we presented a MDD method that targets this category of systems. This method focuses on the critical stages of DCS development. Namely, the specification of system requirements, the choice of OTS modules and fieldbus system, and the validation of the design using specialized real-time analysis tools.

The MDD method that was presented uses UML as support notation, as well as the extensions defined in the UML Profile for Schedulability, Performance and Time Specification [7]. The diagrams shown in this paper constitute a small example that illustrates the use of UML throughout the DCS development process. They apply to CAN-based systems and to real-time analysis tools developed within the Methodes.
project. Nevertheless, the approach that was taken is sufficiently generic to permit a simple adaptation to other fieldbus standards and analysis tools. Present and future work to be undertaken within this project includes the evaluation of the use of this methodology by industrial development teams and the automation of some of the steps in this development process.

References


5. Wolfgang Weck, Jan Bosch, Clemens Szyperski (Eds.), Proceedings of the Second International Workshop on Component–Oriented Programming (WCOP '97)
6. Working Material from the School on Natural Computation, SNAC
8. Reima Suomi, Paul Jackson, Laura Hollmén and Mats Aspnäs (Eds.), Teleworking Environments, Proceedings of the Third International Workshop on Telework
9. Robert Fullér, Fuzzy Reasoning and Fuzzy Optimization
10. Wolfgang Weck, Jan Bosch, Clemens Szyperski (Eds.), Proceedings of the Third International Workshop on Component–Oriented Programming (WCOP '98)
11. Abstracts from the 10th Nordic Workshop on Programming Theory (NWPT'98)
13. Christer Carlsson and Franck Tétard (Eds.), Intelligent Systems and Active DSS, Abstracts of the IFORS SPC-9 Conference
15. Reima Suomi, Jarmo Tähkäpää (Eds.), Proceedings of the Seventh Nordic Combinatorial Conference
16. Christer Carlsson (Editor), The State of the Art of Information System Applications in 2007
17. Christer Carlsson (Editor), Information Systems Day
19. Reima Suomi, Jarmo Tähkäpää (Eds.), Health and Wealth trough Knowledge
20. Johan Lilius, Seppo Virtanen (Eds.), TTA Workshop Notes 2002
21. Mikael Collan, Investment Planning – An Introduction
24. Pirkko Walden, Stina Störling-Sarkkila, Hannu Salmela and Eija H. Karsten (Eds.), ICT and Services: Combining Views from IS and Service Research
25. Timo Järvi and Pekka Reijonen (Eds.), People and Computers: Twenty-one Ways of Looking at Information Systems
26. Tero Harju and Juhani Karhumäki (Eds.), Proceedings of WORDS'03
27. Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.), Turkku Centre for Computer Science, Annual Report 2002