TOWARDS EFFICIENT COMPONENT-BASED SOFTWARE DEVELOPMENT OF DISTRIBUTED EMBEDDED SYSTEMS

Séverine Sentilles

2009
Abstract

The traditional ways of developing embedded systems are pushed to their limits, largely due to the rapid increase of software in these systems. Developers now have difficulties to handle simultaneously all the factors involved in the development such as increasing complexity, limited and shared resources, distribution, timing or dependability issues. These limitations make the development of embedded systems a rather complex and time consuming task, and call for new solutions that can efficiently and predictably cope with the new specifics and requirements of embedded systems to ensure their final quality.

Component-based software engineering is an attractive approach that aims at building software systems out of independent and well-defined pieces of software. This approach has already shown advantages in managing software complexity, and reducing production time while increasing software quality. However, directly applying component-based software engineering principles to embedded system development is not straightforward. It requires a considerable adaptation to fit the specifics of the domain, since guaranteeing the extra-functional aspects, such as real-time concerns, safety-criticality and resource limitations, is essential for the majority of embedded systems.

Arguing that component-based software engineering is suitable for embedded system development, we introduce a component-based approach adjusted for embedded system development. This approach is centered around a dedicated component model, called ProCom, which through its two-layer structure addresses the different concerns that exist at different levels of abstraction. ProCom supports the development of loosely coupled subsystems together with small non-distributed functionalities similar to control loops. To handle the management of important concerns related to functional and extra-functional properties of embedded systems, we have extended ProCom with an attribute framework enabling a smooth integration of existing analysis techniques. We have also demonstrated the feasibility of the approach through a prototype realisation of an integrated development environment.
Résumé
— Abstract in French

Affrontant une rapide et massive introduction de logiciels, le monde des systèmes embarqués est en proie au changement. De ce fait, les méthodes traditionnelles de développement de ces systèmes atteignent leurs limites. Elles ont désormais des difficultés à gérer simultanément tous les paramètres impliqués dans le développement, tel que l’accroissement de la complexité, la limitation et le partage des ressources, la distribution, ainsi que les contraintes temporelles et de fiabilité. Ces limitations rendent le développement particulièrement complexe et coûteux, et requièrent de nouvelles solutions pouvant efficacement et de manière prévisible répondre aux nouveaux besoins des systèmes embarqués afin d’assurer leur qualité finale.

L’ingénierie logicielle basée composants est une approche visant à la construction de systèmes logiciels par l’usage de “briques logicielles” indépendantes et parfaitement caractérisées. Cette approche a déjà démontré des aptitudes pour appréhender la complexité logicielle tout en réduisant les temps de production et maintenant la qualité. Pourtant appliquer directement les principes de l’ingénierie logicielle basée composants au développement de systèmes embarqués n’est pas simple et nécessite une adaptation considérable pour se conformer aux exigences du domaine, telles que la limitation des ressources et les contraintes temps réel et de criticité.

Convaincus que l’ingénierie logicielle basée composants convient au développement des systèmes embarqués, nous introduisons une approche basée composants dédiée au développement de systèmes embarqués. Cette approche s’appuie sur ProCom, un modèle de composants spécifique qui au travers de sa structuration en deux niveaux concerne les propriétés présentes à différents niveaux d’abstractions. ProCom supporte le développement de sous-systèmes
faiblement couplés conjointement avec de petites fonctionnalités non distribuées analogues aux boucles rétroactives. Dans le but d’assurer la gestion des aspects ayant trait aux propriétés fonctionnelles et extra-fonctionnelles, nous avons étendu ProCom au travers d’un “attribute framework” facilitant l’intégration de techniques d’analyses préexistantes. La faisabilité de l’approche est également démontrée via la réalisation d’un prototype d’environnement de développement intégré.
Acknowledgements

Looking back at my past, nothing predestined me to do a thesis and even less in Sweden, a country that I would have never envisaged to live in (“it is too cold up there !!!”). But the course of my life completely changed thanks to Nicolas Belloir, who put his trust in me and always tried to pushed me forward, smoothly enough to manage to make me accept a PhD position at Mälardalen University. I cannot say how much I am thankful to you for this: you are a great friend!

But this adventure would not have been possible nor been as enjoyable either without the intervention of many people. To begin with, I would like to express my gratitude towards two of my supervisors, Ivica Crnkovic and Hans Hansson. Thank you for believing in me and accepting me as a PhD student despite my hesitating Frenglish way of speaking. I am always amazed by your enthusiasm, commitment and above all your inexplicable capacity to work so much. Many thanks also go to my other supervisor, Jan Carlson, for all the fruitful discussions, inputs, reviews, help and guidance every time I needed it. I also want to thank my French supervisors, Frank Barbier and Eric Cariou, who have given me the opportunity to do a so-called “co-tutelle” with the university of Pau.

Many thanks also go to the “Mental Department” that many have tried to enter but few have managed, ProPhs and associated members (Cristina, Stefan/Bob, Hüüs, Tibi, Adnan, Aida, Aneta, Luis, Batu, Farhang, Hongyu, Pasqualina, Juraj, Mikael, Antonio, Ana, Luka, Leo, Marcelo, Jagadish) for all the laughters and great moments during the fika, lunches and travels. You are really great people to work with, and above all great friends. And of course, I don’t forget all the PROGRESS and/or IDT members, Andreas, Damir, Daniel, Lars, Jörgen, Mikael Åkerholm, Radu, Nolte, Markus, Ebbe, Anton, Rikard, Stig, Frank, Paul, Jukka, Sasi, Malin, Gunnar, Åsa, for making life at work and abroad so pleasant!
I would also like to put a special mention to Harriet Ekwall and Monica Wasell not only for continuously helping out on an every-day basis and bringing so much fun in the department but also for all the help they provided me when I arrived in this foreign country and I was totally lost and confused with the administrative procedures. The atmosphere at the department will definitely not be the same without you.

There are also a lot of friends from childhood and university that I really want to thank for having been present for me when I really needed support and good friends: Anouk, Flo, Natacha, Aurel, Cristine, Fafou, Eric, Gael, Sophie, Marie, Pauline, Laure, Aude, Anne-Sophie and Bea. I must say that I am really lucky to have you around.

And last but not least, I would like to thank my parents, grand-parents, cousins (Yan, Aurélie, Cédric, Alex, Lou-Anne), Marie-Françoise, Marie-Paule, Fredo, Nono, and of course Dag and Liv for bringing so much to my life that I cannot express this with words.

Séverine Sentilles
Västerås, November 2009

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the research centre PROGRESS.
List of Publications

Publications Included in the Licentiate Thesis

**Paper A:** *A Classification Framework for Component Models*. Ivica Crnković, Séverine Sentilles, Aneta Vulgarakis, Michel Chaudron. Accepted to IEEE Transactions on Software Engineering (in the process of revision).


---

1The included articles have been reformatted to comply with the licentiate page setting
Additional Publications, not included in the Thesis

Conferences and workshops:


MRTC reports:


To my grandfather
# Contents

## I Thesis

### 1 Introduction

1.1 Motivation ............................................. 3
1.2 Objectives ......................................... 5
1.3 Thesis Overview ..................................... 6

### 2 Background

2.1 Embedded Systems .................................... 11
  2.1.1 Characteristics in Vehicular Domain ........... 13
  2.1.2 Characteristics in Automation Domain .......... 14
2.2 Component-Based Software Engineering .............. 15
  2.2.1 Extra-Functional Properties .................... 16
  2.2.2 The Component-Based Development Process .... 18
  2.2.3 Component-Based Software Engineering for Embedded System Development ........ 19

### 3 Research Summary

3.1 Problem Positioning ................................... 21
3.2 Research Questions ................................... 23
3.3 Research Contribution ................................. 24
  3.3.1 A Classification Framework for Component Models . 25
  3.3.2 Requirements for a Component-Based Approach ... 27
  3.3.3 The ProCom Component Model .................... 29
  3.3.4 Integration of Extra-Functional Properties in Component Models .................. 30
  3.3.5 Prototype Implementation ......................... 32
3.4 Methodology ........................................... 33
### Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Related Work</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Component Models</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Alternative Approaches</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Integrated Development Environment</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Future Work</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Discussions</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Future Work</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>53</td>
</tr>
<tr>
<td>II</td>
<td>Included Papers</td>
<td>61</td>
</tr>
<tr>
<td>6</td>
<td>Paper A: A Classification Framework for Component Models</td>
<td>63</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>The Classification Framework</td>
<td>67</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Lifecycle</td>
<td>68</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The Constructs</td>
<td>71</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Extra-Functional Properties</td>
<td>75</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Domains</td>
<td>79</td>
</tr>
<tr>
<td>6.2.5</td>
<td>The Classification Overview</td>
<td>80</td>
</tr>
<tr>
<td>6.3</td>
<td>Survey of Component Models</td>
<td>82</td>
</tr>
<tr>
<td>6.3.1</td>
<td>“Almost” Component Models</td>
<td>82</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Component Models</td>
<td>83</td>
</tr>
<tr>
<td>6.4</td>
<td>The Comparison Framework</td>
<td>84</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Lifecycle Classification</td>
<td>84</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Constructs Classification</td>
<td>86</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Extra-Functional Properties Classification</td>
<td>89</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Domains Classification</td>
<td>91</td>
</tr>
<tr>
<td>6.5</td>
<td>Related Work</td>
<td>92</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusion</td>
<td>93</td>
</tr>
<tr>
<td>6.7</td>
<td>Appendix — Survey of Component Models</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>103</td>
</tr>
<tr>
<td>7</td>
<td>Paper B: A Component Model Family for Vehicular Embedded Systems</td>
<td>109</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>111</td>
</tr>
<tr>
<td>7.2</td>
<td>Motivating Example</td>
<td>113</td>
</tr>
</tbody>
</table>
7.3 The **PROGRESS** Approach ........................................ 115
7.4 Towards CBD in Vehicular Systems ............................... 117
  7.4.1 From Abstract to Concrete .................................... 117
  7.4.2 Component Granularity ....................................... 120
7.5 Conceptual Component Model Family ............................. 120
7.6 Realization of the Proposed Component Model Family ............ 122
7.7 Related Work ..................................................... 124
7.8 Conclusion ....................................................... 125
Bibliography .......................................................... 127

8 **Paper C:**
   A Component Model for Control-Intensive Distributed Embedded
   Systems ................................................................. 129
   8.1 Introduction .................................................... 131
   8.2 The ProCom Two Layer Component Model ....................... 132
      8.2.1 ProSys — the Upper Layer ................................. 132
      8.2.2 ProSave — the Lower Layer ............................... 133
      8.2.3 Integration of Layers — Combining ProSave and
           ProSys .......................................................... 136
   8.3 Example .......................................................... 137
   8.4 Conclusions ..................................................... 138
Bibliography .......................................................... 141

9 **Paper D:**
   Integration of Extra-Functional Properties in Component Models 143
   9.1 Introduction .................................................... 145
   9.2 Annotation of Attributes in Component Models .................. 146
      9.2.1 Attributes in a Component Model ........................ 147
      9.2.2 Attribute Definition ...................................... 147
      9.2.3 Attribute Type ............................................. 149
      9.2.4 Attribute Data ............................................. 150
      9.2.5 Multiple Attribute Values ................................ 151
      9.2.6 Attribute Value Metadata ................................ 152
      9.2.7 Validity Conditions of Attribute Values ................ 152
   9.3 Attribute Composition .......................................... 154
   9.4 Attribute Configuration and Selection .......................... 155
   9.5 A Prototype for ProCom and the **PROGRESS** IDE ............ 158
   9.6 Related Work ..................................................... 160
   9.7 Discussion ....................................................... 162
10 Paper E:  
Save-IDE – A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems  
10.1 Introduction ........................................... 173  
10.2 Software Development Process .................... 174  
10.3 Component-Based Design .......................... 176  
10.4 Analysis .............................................. 178  
10.5 Synthesis ............................................. 179  
10.6 Conclusion ........................................... 180  
Bibliography ............................................. 180
Chapter 1

Introduction

Development of embedded software is a complex process significantly influenced by human factors — from the way the software is designed to the errors introduced during the implementation phase, and some of which remain in the product after release. Yet, providing the appropriate functionality is not sufficient anymore, the product has also to be produced in an efficient way and be trustworthy! This is the main concern of this thesis, which investigates methods and techniques to improve software development by helping guaranteeing that the delivered products will meet stringent quality requirements like the ones that are inherent to a lot of embedded systems.

1.1 Motivation

Having a suitable and efficient development is an essential concern when developing safety-critical systems for a variety of domains such as vehicular, automation, telecommunication, healthcare, etc. since any malfunction of these systems may have severe consequences ranging from financial losses (e.g. costs for recall of non-conformity products) to more harmful effects (e.g. injuries to users or in the most extreme cases users’ death). Along with their traditional mechanical functionalities, e.g. a combustion engine or mechanical brakes in a car, these products also contain more and more software functionalities, such as for instance an anti-lock braking system or an electronic-stability control unit in a car. This means that similarly to what is done for the mechanical elements, software parts require to be meticulously developed and verified to ensure the
essential quality of the delivered products: their dependability. That is to say that their reactions to events are the ones expected in the adequate amount of time. Their development must hence support thorough analysis and tests, and push these activities even further compared to what can be found in traditional software engineering.

Software functionalities in those types of product are provided through special-purpose built-in computers, called embedded systems, which are tailored to perform a specific task by combination of software and hardware. Another fundamental characteristic of those systems is that they often have to function under severe resource limitations in terms of memory, bandwidth and energy, and even sometimes under difficult environmental conditions (e.g. heat, dust, constant vibrations). Even though the introduction of software functionalities, sometimes as replacement for hardware ones, offers tremendous opportunities, it also considerably increases the software complexity. For example, in the vehicular domain, the demand for additional software is constantly increasing [1]. Consequently in this particular domain, the traditional solution of decomposing the required functionalities into subsystems that are realised by dedicated computing units using their own microcontroller does not scale any more. Instead, there is a need to put several subsystems on one physical unit, which implies that resources must be shared between subsystems. Another aspect of this increasing complexity is distribution, as systems also often tend to be designed as distributed systems communicating over a dedicated network such as a CAN-bus [2] or a LIN-bus [3] in a car. The interdependence of these concerns together with the need for thorough verification of the system make the development of embedded systems rather difficult and time-demanding.

A promising solution for the development of distributed embedded systems lies in the adoption of a Component-Based Development (CBD) approach facilitating the different types of analysis needed. The CBD approach has the goal to increase efficiency in software development by:

- reusing already existing solution encapsulated in well-defined entities (components);
- building systems by composition of those entities (both from a functional and extra-functional point of view); and
- clearly separating component development from system development.
Several features proposed in the CBD approach are of high interest in the development of distributed embedded systems, such as:

- complexity management;
- increased productivity;
- higher quality;
- shorter time-to-market;
- lower maintenance costs; and
- reusability.

However, despite those appealing aspects and its establishment as an acknowledged approach for software development, notably for desktop or business applications [4], CBD still struggles to really break through for embedded system development. For a better acceptance in this domain, the main challenge is to deal with both complexity and functional requirements on one hand, and on the other hand to deal with the specifics related to embedded systems and their particular development needs — including support for extra-functional requirements, strong dependence on hardware, distribution, timing issues and limited resources. Still, several approaches to use CBD in embedded systems can be found, such as AUTOSAR [5], BlueArX [6, 7], SaveCCM [8], Rubus [9], Koala [10] and Pecos [11]. More detailed information about the different component models for embedded systems can be found in Chapter 4. However, even if all these approaches were successful in solving particular aspects of the development process, an approach that supports the use of components throughout the whole development process — from early design specification to system deployment and synthesis — and provides grounds for the various type of required analysis is still needed. This is the main concern of this thesis.

1.2 Objectives

The main purpose of this licentiate thesis is to propose solutions towards establishing an efficient software development of distributed embedded systems that can ensure the quality of the delivered products. Assuming that the principles advocated in CBD are also applicable for developing distributed embedded systems, this thesis discusses how to suitably accommodate the specifics
Chapter 1. Introduction

of “traditional” embedded system development with component-based development and, then how to integrate and manage extra-functional properties in the development to ensure the quality of the final product. This thesis also focuses on determining the required engineering practices and tools to efficiently support the composition theories which have been proposed.

Concretely, in this thesis we propose a component-based approach for distributed embedded systems supported by the specification of a dedicated component model. This component model is endowed with suitable characteristics, properties, and features to efficiently support the management of the specific concerns of embedded system domain, in particular the integration and management of extra-functional properties as means to bridge analysis in the development process. The approach is illustrated through the realisation of an integrated development environment.

1.3 Thesis Overview

This thesis is organized in two distinct parts. The first part gives a summary of the research; Chapter 2 introduces technical concepts used throughout the thesis, Chapter 3 describes the research which has been conducted in presenting the motivation for the research, the research questions, the research contributions and the research methodology. Chapter 4 introduces the related work, and Chapter 5 concludes and presents the future work.

The second part consists of a collection of peer-reviewed journal, conference and workshop papers, presented below, contributing to the research results.

Ivica Crnković, Séverine Sentilles, Aneta Vulgarakis, Michel Chaudron (Technical University Eindhoven). Accepted to IEEE Transactions on Software Engineering (in the process of revision).

Summary
Based on the study of a number of component models which have been developed in the last decades, this paper provides a Component Model Classification Framework which identifies and discusses the basic principles of component models. Through the utilization of this classification framework, this paper also pinpoints differences between component models and identifies common characteristics shared by some component models developed for a similar domain, such as embedded systems.
My contributions
This paper has been written with an equal contribution of the first three authors concerning the analysis of the selected subset of component models, the specification of the classification framework and the iterative process to refine the framework. All the co-authors contributed with discussions, reviews and suggestions. Personally, I contributed to the paper with the initial idea of classifying component models and during the work, I was more specifically in charge of the work around the constructs dimension of the framework and the related work. The classification framework was developed in several iterations, including discussions with CBSE experts from both academia and industry.


Summary
This paper describes the high-level views which have guided us towards the elaboration of ProCom (a component model for the design and development of distributed embedded systems; see Paper C), namely the needs for (i) having several component concepts corresponding to the different levels of abstraction considered (big components/small components); (ii) the ability to deal simultaneously with components in different state such as early-design components or fully implemented reused component (abstract components/concrete components); (iii) managing the strong coupling with the target platforms; and (iv) having a component model ready to be enhanced with various analysis.

My contributions
This paper is the outcome of an equal contribution of all authors. More specifically I contributed to this paper by participating in the discussions concerning the development process, the discussions with the domain experts to collect information on their needs and by influencing some of the decisions through my parallel work on the realization of an integrated development environment, called Save-IDE, for the SaveCCM component model. The work summarized in this paper is the result of an iterative process starting with the knowledge gained from the SaveCCT approach and involving many other members of the PROGRESS project, who contributed with valuable discussions and inputs for the proposed ideas.
Paper C: A Component Model for
Control-Intensive Distributed Embedded Systems.

Summary
In this paper, we present the Progress component model (ProCom) for the design and development of control-intensive distributed embedded systems. The particularity of this component model lays in the existence of two layers designed to efficiently cope with the different design paradigms which exists on different abstraction levels in the vehicular domain. Moreover through the utilization of a component-based development, the aim is to decrease the complexity in design and provide a ground for analyzing the components and predict their properties, such as resource consumption and timing behaviour.

My contributions
This paper is strongly related to Paper B and is also the outcome of an equal contribution of all authors. More specifically I contributed to this paper in participating in the discussions concerning the development process, the discussion with the domain expert to collect information on their needs and influencing some of the decisions through my parallel work on the realization of an integrated development environment, called Save-IDE, for the SaveCCM component model. Similarly to the work presented in the previous paper, the work around the ProCom component model started with an attempt to refine SaveCCM and has been carried out in several iterations involving many PROGRESS members.

Paper D: Integration of

Summary
This paper looks at the diversity that exists in specifying extra-functional property (e.g. timing, behaviour or resource properties) and, proposes a way
to integrate and systematically manage extra-functional properties within component models. This is done with the main objective to provide an efficient support, possibly automated, for analysing selected properties. In this paper, a format for attribute specification is proposed, discussed and analyzed and the approach is exemplified through its integration both in the ProCom component model and its integrated development environment.

*My contributions*

I was the main author and driver of this paper and contributed with the attribute definition for extra-functional properties, the literature survey and the supervision of a master student leading to a prototype implementation based on preliminary ideas. All the co-authors contributed with valuable discussions, advices and suggestions all along the work.


*Summary*

This demo paper presents an integrated development environment for the development of predictable component-based embedded systems. Save-IDE supports efficient development of dependable embedded systems by providing tools for design of embedded software systems using exclusively the SaveCCM component model, formal specification and analysis of component and system behaviours already in early development phases, and a fully automated transformation of the system of components into an executable image.

*My contributions*

I was the main driver of this paper and I have contributed to it in being involved in the realization of the environment (specification, implementation) and in the writing of most parts of the paper. More concretely concerning the realization, I was a member of the developing team with a responsibility for the design part, including the design of the underlying metamodel, and the development of the design tools.
Chapter 2

Background

This section briefly introduces important technical concepts used throughout the remainder of this thesis. It provides an introduction to embedded systems and their characteristics (Section 2.1) and to component-based software engineering (Section 2.2). However, for more information on embedded systems, we refer to [12] or [13], and for details on component principles and technologies to [4], [14] or [15].

2.1 Embedded Systems

Embedded systems have managed to spread rapidly over the past few decades to be virtually in any kind of modern appliances such as digital watches, set-top boxes, mp3-players, washing-machines, mobile telephones, cars, aircrafts, forest machines and many more. Because of this, a uniform definition covering this diversity is difficult to pinpoint and therefore there is currently no unique definition of what they are. For example, IEEE states that “an embedded computer system is a computer system that is part of a larger system and performs some of the requirements of that system”. In this thesis, we denote by embedded system a special-purpose computer built into a larger device and tailored to perform a specific task by combination of software and hardware. In contrast to general purpose computers, embedded systems are (i) reactive systems closely integrated into the environment with which they interact through sensors and actuators, (ii) often strongly resource-constrained in terms of memory, bandwidth and energy and, for some of them (iii) possibly confronted to harsh environmental conditions enduring dust, vibrations, heat, etc.
The close interconnection of embedded systems with their surrounding environment and their ability to directly impact on this environment leads to another characteristic shared by many embedded systems: their safety-critical nature. Accordingly to prevent any malfunction which could lead to a problematic situation ranging from financial losses (e.g. costs for non-conform products recall) to more dramatic ones (e.g. device loss, users’ injuries or in the most extreme cases users’ death), they have to react in well-specified ways and be highly dependable. As mentioned in Laprie’s definition [16], dependability of a system is the quality of the delivered service such that a user can justifiably placed reliance on this service. In particular, dependability is expressed in terms of safety (i.e. the failure of the system must be harmless), maintainability (probability that a failure can be fixed within a predefined amount of time), reliability (probability that the system will not failed) and availability (probability that the system is working and accessible) among others.

Also, many embedded systems have to observe real-time constraints, which means that they must react correctly to events in a given interval in time. When all the timing requirements must strictly be ensured, embedded systems are called hard real-time systems whereas soft real-time systems are more flexible towards the timing bounds and can tolerate to occasionally exceed them. One popular example to illustrate this strong interdependence between real-time and dependability issue is the one of a car airbag. In case of accident, the airbag has to inflate suitably at a particular point in time, otherwise it is useless for saving the driver’s life. One major issue in dealing with safety-critical real-time embedded system is therefore to ensure that the system always behaves correctly.

It is worth noting that the great diversity of devices containing embedded systems makes the boundaries between what it is considered to be embedded systems and what is not particularly unclear. Many devices share characteristics with embedded systems without necessarily been considered as such. Notebooks, laptop or personal digital assistants are few examples of devices in the grey zone of the definition of embedded systems: they are resources-constrained and possibly integrated into the real world through various equipment such as GPS but they are still regarded as “bigger” than archetypical embedded systems. Conversely although containing desktop-like software and means to interact with users, others devices such as control-system for robots are still considered as embedded systems.

Since present in many different devices and forming a heterogeneous class of applications, complexity and requirements of embedded systems vary from one application domain to another. The following subsections 2.1.1 and 2.1.2
detail the characteristics of embedded systems and the current state of practice of their development for the domains this thesis is more particularly concerned with.

2.1.1 Characteristics in Vehicular Domain

Nowadays the added-value in high-end models of cars is generated mainly by the integration of new electronic features that are intended to optimize the utilization costs of the vehicle (e.g. lower fuel consumption), or to improve the user’s comfort or safety. According to [17] in 2006, 20% of the value of each car was due to embedded electronics and this was expected to increase to 36% in 2009. This involves features such as airbag control system, anti-braking system, engine control system, electronic stability control system, global positioning system, door locking system, air-conditioning system and many more. More generally speaking, these features concern control, infotainment (i.e. information and entertainment) and diagnosis systems.

To realize these systems, the physical system architecture of a modern vehicle consists of large number of computational nodes called Electronic Control Units (ECUs) that are distributed all over the car and connected by several different communication networks, principally CAN [2], LIN [3], MOST [18] or Flex Ray [19] buses. Traditionally in the vehicular domain, one functionality corresponds to one ECU and its development is characterized by the extensive use of sub-contractors. After having received a specification from the car manufacturer, the sub-contractors design both the software and the hardware of the subsystem to deliver. Consequently, sub-contractors are involved in the addition of mechanical parts to the system enforcing a strong coupling between the software and the hardware parts. In this way of developing embedded systems, the test of the overall system is realized really late in the development process after the integration of all the subsystems, which is extremely costly.

The rapid introduction of software functionalities in vehicles challenges significantly the current development practice in the vehicular domain since it induces to find solutions to elaborate a design as close as possible to an optimal system design (both with respect to cost and resources usage) that can provide the desired functionality with a sufficient level of dependability. Whereas car manufacturers strive for low production costs since each car model is manufactured in large quantities, the biggest costs — up to 40% of the production costs [20] — resides in software and electronics costs. Lowering these costs requires dealing with the tight coupling which exists between the software and hardware parts, distribute functionality across several ECUs which implies an
increase of the interdependencies and connections between ECUs (for example a “simple” interior lightning system can involve up to ten ECUs distributed all over the car), allocate several functionalities to a same ECU to optimize the resource utilization, and manage the growing complexity.

2.1.2 Characteristics in Automation Domain

Industrial automation has pushed the mechanization one step further in intensively using embedded systems — in particular programmable logic controllers (PLCs), a type of control systems. The motivation behind this is to have better control over the production processes and optimize them to provide high-quality and reliable products by minimizing material, costs, energy waste and human intervention.

In this particular domain, embedded systems consist of sensors and actuators connected with an open and standardized field bus to, possibly distributed, control systems. In difference to other embedded system architectures, they are used conjointly with end-user technologies that serve as interfaces between human and machine to control and operate the system as for example the temperature in a pipe, the pressure of a valve or the arm of a production robot.

Other similarities exist with embedded systems present in the vehicular domain. In particular, many applications share the safety-critical, real-time and resource requirements of the vehicular domain. In both domains, embedded systems are manufactured in large volume and their development is often based on control-theory.

Aside from these similarities, principal differences also exist. The presence of a human-machine interface constitutes a major difference. It implies a need for a seamless integration and higher interoperability of embedded systems with “more advanced” technologies which are not necessarily real-time constrained. Also these embedded systems are developed to be present in long-life products which need to be reconfigured or adapted to switch easily from manufacturing one product to another without having to completely rebuild the production lines. This means that embedded systems for automation domain must be easily portable to a new hardware and cope with legacy systems.

Contrary to the automotive domain, which is relatively new to software engineering methods, the automation domain has a strong tradition in software engineering. Many embedded systems are developed in following some standards, such as IEC-61131-3 [21].
2.2 Component-Based Software Engineering

Building products out of well-defined and standardised parts is an old engineering practice that can be traced back to Henry Ford and the mechanisation era. Many advantages emerge from this way of developing products: short time-to-market, lower maintenance time and costs, and reusability of the pieces across different products. Inspired by the successes engendered in industries and envisioning similar benefits, Component-Based Software Engineering (CBSE) aims at applying this development practice to software development. Following this standpoint, the construction (resp. decomposition) of software systems must be based on independent and well-defined pieces of software, called components.

However, whereas in other engineering disciplines, the concept of components is intuitively graspable since it is generally a physical object that can be manipulated, directly transferring this notion to software engineering is not straightforward. The fuzziness around the notion of component is put in evidence by the number of definitions that exists today. In [15], no less than fifteen definitions are compared to each other. Out of those definitions, probably the most commonly acknowledged one is from Szyperski [22] which highlights some fundamental characteristics of a component: communication through well-specified interfaces only, composability and reusability by third party. This definition states that:

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party”

As pointed out by this definition, an important characteristic of a component specification is its interfaces. An interface is the specification of an access point to the component’s functionality described as a collection of available operations. A distinction between two types of interfaces exists. A required interface expresses the functionality requested by the component to function correctly whereas conversely, a provided interface describes the functionality offered by the component. In that sense, interfaces are used for enabling interaction with other components and external environment, and to compose or “link” components together.

In addition to the concepts of component and interface, a fundamental notion is the one of component model. A component model defines all the characteristics and constraints that the components and the supporting component
framework — i.e. the tools for manipulating the components — must satisfy. A component model is concerned with providing (i) rules for the specification of component properties and (ii) rules and mechanisms for component composition, including the composition rules for properties. In that sense, a component model provides the cornerstone of standardization for software development. For instance, Heineman and Councill [14] propose a component definition in regards to a component model:

“A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard”.

In applying those concepts, component-based software engineering has already been proven to be successfully used in domains where no strong timing-requirements are needed such as information, service-oriented or desktop systems [4]. This success is highlighted by the proliferation of component models which exist today (see Paper A in which twenty-four component models are compared).

### 2.2.1 Extra-Functional Properties

For many years, component-based software engineering has essentially focused on providing methods and techniques to support the development of software functionalities in an efficient way. Yet, for certain types of applications such as dependable, real-time or embedded systems, other factors are as important for a smooth running of the system as the functionality itself. These factors describe the non behavioural aspects of a system, capturing the properties and constraints under which that system must operate [23]. For example, they relate to the capability of the system in terms of reliability, safety, security, maintainability, accuracy, compliance to a standard, resource consumption, and timing properties, among many others. These factors can be found under several denominations, the most common ones being non-functional properties, extra-functional properties, quality attributes or simply attributes. In this thesis, we refer to these factors through the use any of these terms indifferently.

As a consequence of the little attention to these factors, few component models actually provide support for specification and management of extra-functional properties. This is especially true for widespread general-purpose component models such as COM [24], CCM [25], .NET [26] or EJB [27]. Besides, when this support is available, it takes different forms — unlike behavioural factors for which the well-established solution of embodying the
functionalities into the interfaces exists. First, this support can be provided at component-level through additional interfaces, called introspective or analytical interfaces. Used at design-time, these interfaces allow for early analysis of the component or the system, whereas their utilisation at run-time enables mechanism such as monitoring. Another way of supporting extra-functional properties is to provide annotations through name-value pairs specifications. The last way is to use a dedicated language or mechanism outside the component model itself.

Besides providing means for their specification, dealing with extra-functional properties with respect to the CBSE principles raise challenges related to composability or reusability issues. Similarly to the composability challenges for components, we would also like to be able to reason about their composition, in that sense that the values of a property \( P \) of a compound element \( A \) is the result of the composition of the values of the inner components \( C_1 \) and \( C_2 \):

\[
A = C_1 \circ C_2 \Rightarrow P(A) = P(C_1) \circ P(C_2)
\]

However, as described in [28], few properties are directly composable in following that principle. The value of many extra-functional properties is influenced by other factors such as the software architecture, other properties, the usage profiles and/or the current state of the environment.

Dealing with extra-functional properties in the context of component-based software engineering also raises the issue of reusability since it is one of the cornerstone concept around which component-based approach is built. Indeed, when a component is reused in different applications or contexts, the extra-functional properties associated to this component must also be reusable, in that sense that their values are still accurate in the current setting. However, many property values depend upon information outside the component model itself. Therefore in order to reuse the extra-functional properties, means to evaluate the conditions under which the value is correct are required. A typical example is a worst-case execution time, which requires information about the compiler used to generate the executable code but also about the target platform specification such as the type of memory, processor or the presence of caches, among many other factors.
2.2.2 The Component-Based Development Process

The specific aspect of developing software consistent with the CBSE principles is based on a strict separation between component development and system development (with components). Both processes can follow the traditional “Requirement, Specification, Implementation, and Verification” phases whether, for instance, in a waterfall or V-model form. However, due to the presence of components characteristic features emerge. Both processes and their interaction are illustrated in Figure 2.1.

Figure 2.1: Component-based development process overview.

Starting normally with an elicitation of system requirements, the system development takes immediately advantage of the presence of previously developed components which are stored in a component repository. Based on the knowledge and identification of a set of component candidates that potentially fit the requirements, system requirements are broken down into component
requirements and accordingly, a system specification is built with its corresponding component specifications. Whereas the components that do not completely fit the specifics of the current design are adapted, the requirements and specification of the non-already implemented components are forwarded to the component development process to be developed. Once all components have been implemented and individually tested against their requirements, they are integrated together to form the final system. This integration is then verified and validated against the system requirements, both with regards to functional and extra-functional aspects.

As for the component development process, the steps are generally quite comparable to the ones found in traditional software development. Based on requirements and specification coming from system development, components are implemented and tested against these requirements. When the components meet their individual requirements, they are then delivered to be integrated during the system development and/or stored in repository as candidate for future reuse. However the component development process also aims at building components satisfying requirements not issued from system development but extracted to realize more generic components that can be used in many different contexts. This way of developing component is more difficult since it requires to envisage all possible contexts in which the component will be used. This generates components that are bigger than custom-made components since they need to fit more usage contexts. This introduces challenges for embedded system development since it requires efficient components.

2.2.3 Component-Based Software Engineering for Embedded System Development

Contrary to other domains in which component-based software engineering have proven to be successfully used for common software development (desktop, business, internet or entertainment applications), CBSE has still difficulties to really breakthrough for the development of embedded systems. Indeed, most of the existing general-purpose component technologies have been developed with little consideration to factors that are of high important for embedded systems such as their resource limitations, timing properties or safety-criticality.

The mismatch between the requirements for developing traditional PC applications and the ones for embedded systems hinder a straightforward transfer of these component-based technologies from one domain to another. In particular, the widespread component technologies such as EJB [27], .NET [26], COM [24] or CCM [25] do not sufficiently address these fundamental require-
ments and as a result are not that suitable for embedded systems development. They present some major drawbacks in being heavyweight, complex and generating some significant overhead on the target platform. As a consequence and as pointed in [29], there is still no widely used component technology standard really suitable for embedded systems.

However, the principles and promising advantages brought out by CBSE have drawn a general attention towards fostering the use of component models for embedded system development. Several recent initiatives to provide standards based on component-based principles as well as the elaboration in the recent years of a number of component models dedicated to embedded systems reflect such a change. Some of these dedicated component models are KOALA [10], RUBUS [9], BlueArX [6, 7], SaveCCM [8], IEC-61131 [21] and AUTOSAR [5]. More details about these component models can be found in Paper A.
Chapter 3

Research Summary

In this chapter, we describe the research performed. We first state the problem that this thesis addresses, then formulate the research questions, summarize the research results which contribute to answering those research questions, and present the used research methodology.

3.1 Problem Positioning

Facing a growing demand to integrate more and more software functionalities, the traditional development methods for embedded systems are showing their limits. They have difficulty to efficiently cope with the resulting problems, namely increasing complexity, distribution, stringent resource limitations, a strong coupling between software and hardware, timing properties, safety-critical issues, etc. An important challenge is thus to propose development methods supporting those new requirements to facilitate embedded software development and ensure the quality and the dependability of the delivered products.

Motivated by the need for solutions, the main challenge that this thesis aims at addressing can be formulated by the following question:

*How can distributed embedded systems be developed in a predictable and efficient way while following the CBSE principles?*

Otherwise stated, this means that this thesis aims at clarifying what are the important characteristics that the development of embedded systems requires and
determining how to adapt the prerequisite of CBSE to suitably handle these characteristics. In particular, this can be seen as developing a suitable component technology which aims at providing support to address the embedded system requirements.

Therefore the main research objective of this thesis is to propose concepts, approaches, and techniques concerned with the elaboration of an efficient component-based software development for distributed embedded systems, covering the development process stages (from early design to system deployment and synthesis) as well as enabling reusability and various types of analysis. It also looks at determining the needed engineering practices and tools to support the theories which have been proposed. However, this thesis is not interested in distribution primarily, and does not aim at providing new distribution architecture or communication protocols. Distribution is only considered for the sole purpose that subsystems can be distributed across the architecture and communicate through dedicated networks, as is the case in the vehicular domain for instance.

Besides, other factors, outside the scope of this thesis, need also to be investigated to foster the usage of CBD and improve its efficiency for embedded system development. This is the case of development processes, businesses processes, or devising suitable analysis theories complying with the component-based theories.

The problem envisaged in this thesis is quite broad. In order to reduce its scope, we have worked under assumptions issued from a previous work done at MDH on the SaveCCT development approach ([8], [30]). This work has shown the value of having a restricted component model to help in the analysability of the system already in the design phase. Accordingly, we have considered the following research assumptions:

- A specific component model for distributed embedded system, with a precise semantic is needed;

- Composition theories alone are not enough and require the existence of technologies which include appropriate tool support;

- Introducing verification of extra-functional properties in the early phases of the development process is necessary.


3.2 Research Questions

In order to reduce the scope of the research and define a direction to provide answers to it, three research questions, hereafter described, are stated. The answers to these research questions will unveil important aspects contributing to answering the main question.

Research question 1

*What are the suitable characteristics of a component model to efficiently support software design of distributed embedded systems?*

Through this research question, the purpose is (i) to explore and identify important needs in the development of distributed embedded systems, focusing more specifically on the design phase while keeping in mind that a component-based approach is intended, and (ii) to adapt an existing (or propose a new) component model with suitable characteristics, properties and features to provide a solution to these needs.

In order to provide an answer to this question, we first study the development process of distributed embedded systems with the aim to identify concerns that need to be addressed by the component model. The second step is to investigate which kinds of component models exist nowadays, what their characteristics and their domain of applicability are, and if they can be used in the context of this research. Finally, based on the previous results and the work assumptions, the decision of adapting an existing component model or proposing a new one has to be taken.

Research question 2

*How to provide efficient integration support for management of functional and extra-functional properties within a component model?*

This research question aims mainly at the predictability aspect needed in the development of distributed embedded systems in order to provide the necessary quality of the system to be developed. In that respect, this research question focuses on determining a way to enhance the component model to provide the necessary grounds to efficiently support the analysis of important properties. Since various types of information need to be created and used as a basis for taking decision and/or analysing the system under development, it is important to have means to identify, specify, and locate these pieces of information.
To answer this research question, we have (i) identified and described a set of properties which are suitable in the context of the development of distributed embedded systems; (ii) identified to what component model entities (components, interfaces, bindings, etc.) those properties relate; (iii) enhanced the proposed component model to support the management of those properties.

Research question 3

*How to build an integrated development environment encapsulating suitable models and technologies to efficiently support component-based development of software for embedded systems?*

This research question addresses the practical needs required to efficiently support the development of embedded systems. With this research question, the main goal is to develop a prototype and evaluate the feasibility of the approach.

### 3.3 Research Contribution

The contribution presented in this thesis is the outcome of a set of results contributing in the elaboration of efficient component-based software development enabling the development of predictable distributed embedded systems. In this respect, the contributions of this thesis are the following:

- a classification framework for component models;
- requirements for a domain specific component-based approach for embedded systems;
- a component model for distributed embedded systems;
- a method to integrate and manage extra-functional properties within component models; and
- a prototype implementation of an integrated development environment that implements the overall approach.

Figure 3.1 illustrates how these research results fit together to form the overall contribution of this thesis. Through literature surveys and interviews, challenges and needs in the current development methods for embedded systems (Paper B) as well as requirements for merging of CBSE principles with embedded systems development (Paper A and B) have been explored. Based
on the findings, several methods to improve the component-based software development for distributed embedded systems have been proposed (Paper B, C and D). Meanwhile, a prototype implementation (Paper E) based on a SaveCCT has been developed to demonstrate the feasibility, advantages and drawbacks of combining CBSE design with various analysis and deployment techniques to produce embedded systems. The work on this prototype implementation has also influenced the proposed methods.

Next, a brief overview of these research results is given. More details can be found in the included papers in the second part of this thesis.

### 3.3.1 A Classification Framework for Component Models

The idea behind the elaboration of the component model classification framework is to study component-based software engineering state-of-the-art to extract the key principles of the area and analyse their integration within existing component models. Through the utilisation of this framework, principal similarities and differences between component models can be identified as well as their conformance to the CBSE basic principles.

After a thorough study of CBSE state-of-the-art including many component model descriptions and existing classifications of component models, architec-
ture description languages and quality attributes, the following four dimensions have been chosen as main criteria to describe different facets of component models:

1. **Lifecycle**, which identifies the support provided (explicitly or implicitly) by the component models, in certain points of the lifecycle of components.

2. **Constructs**, which identifies (i) the component interface used for the interaction with other components and external environment, (ii) the means of component binding and, (iii) the interaction capabilities.

3. **Extra-functional properties**, which identifies specifications of different property values, and means for their management and composition.

4. **Domains**, which shows in which application and business domains the component models are used or supposed to be used.

Each dimension has then been refined into several aspects and the framework has been populated with more than twenty component models from various domains. The overall classification scheme as well as more details concerning the classification framework can be found in Paper A.

In addition to allow performing a raw comparison between component models by identifying their common characteristics and differences, such a classification framework can also be used for other purposes. In particular, it can serve as a basis to select a component model according to criteria such as the presence of a support for a specific extra-functional property, its implementation language or the support for all the development phases. Ultimately, it could also help in the convergence towards a standardization of main characteristics of component models.

The use of the classification framework in the context of this thesis constitutes the first step towards the identification of suitable characteristics of component models dedicated to embedded system development and a support to eventually determine if an already existing component model could be reused. From the analysis of the classification framework with regards to component models dedicated to embedded systems development, the following characteristics can be extracted as suitable for component models for embedded systems (assuming that the majority is always right).

- communication style: synchronous pipe & filter
- implementation language: C (or C++)
In comparison to general purpose component models, dedicated component models are more concerned with dealing with extra-functional properties and provide support to manage certain type of properties (often timing and resource usage).

### 3.3.2 Requirements for a Component-Based Approach

Based on an evaluation of embedded system requirements and their development needs, the main objective with this work is to (i) establish concepts and requirements suitable for a component-based approach for distributed embedded systems, and (ii) characterise the component model underlying it.

As pointed out in Section 2, a key characteristic of embedded system development is the importance of producing reliable embedded systems in an efficient way. In our view, this requires the provision of a fully integrated approach managing traceability and dependencies between the artefacts generated during the development process such as source code files, models of entities, analysis results, design variants, etc. as well as providing means for various analysis techniques throughout the whole development process. Following this standpoint, a suitable component-based approach for distributed embedded systems (see Paper B) should cover the whole development process starting from a vague specification of the system based on early requirements up to its final and precise specification and implementation ready to be synthesized and deployed. It should also be centered around a unified notion of components as a first-class entity gathering requirements, documentation, source code, various models, predicted and experimentally measured values, etc. and, (iii) improve the predictability of the developed systems by easily enabling various types of analysis, storing and managing the artefacts needed and/or produced by these analysis throughout the development process.

Merging embedded system requirements with a holistic component-based approach throughout the whole development raises the need to cope simultaneously with:

- the coexistence of different abstraction levels,
- the different concerns at different granularity levels,
- platform dependence,
- the need to integrate various analysis techniques throughout the whole development, and
- the need to foster reuse.

Our solution to address these different concerns lays in a conceptual component model composed of two dimensions. The first dimension is the abstraction level (the abstract-to-concrete scale in Figure 3.2), which describes the successive refinement from a rough sketch of a component to its final realisation consisting of source code, detailed timing and resource models for instance. The second dimension expresses the granularity level, i.e. the complexity and size of the components to realise, and is represented by the big-to-small scale in Figure 3.2. For example, an anti-lock braking system (ABS) that constantly adapts the brake pressure in accordance with the wheel speed to prevent wheel skidding while braking belongs to the big part of the scale. On the other hand, a brake force controller which task is only to monitor and adjust the pressure in a brake belongs to the small part of the scale. As illustrated in Figure 3.2, a component can be in different abstraction levels.

Figure 3.2: Proposed conceptual component model.

This work has set the conceptual foundations which guided us towards the elaboration of ProCom, the component model for control-intensive distributed embedded systems described briefly in the next section.
3.3 The ProCom Component Model

With this work, the aim is to specify a component model dedicated to the
development of control-intensive distributed embedded systems for the vehicular
and automation domains primarily. This component model is intended to pro-
vide the cornerstone of the integrated component-based approach described in
Section 3.3.2 and therefore must address the concerns identified above. Taking
these concerns into account, the ProCom component model has been devel-
oped.

To address the first concern, namely the different abstraction levels, Pro-
Com proposes to specify components as black boxes in the early design stage.
In this particular case, a black box component is a component with its internal
content is hidden because it has not been decided yet. During the development,
it can be decided that the component will be a composite component built out
of subcomponents or a primitive component realized through source code. This
means that information is gradually associated with the component, including
adding detailed models for specifying its internal structure, its behaviour, its re-
source usage and finally, with the provision of its source code, the component
is transformed from a abstract black box component to a concrete component.
In that sense, components are viewed as units of design, implementation and
reuse. They can be developed independently, stored in repositories and reused
in multiple applications. To that purpose, ProCom is centered around a unified
notion for components which are considered as a collection gathering all the
information needed and/or specified at different points of time of the develop-
ment process.

The different concerns that exist at different levels of granularity is ad-
dressed through a partitioning of ProCom in two distinct layers of hierarchical
component models. In addition to propose different support to handle these dif-
dferent concerns, the layers differ in terms of architectural styles and associated
semantics for the components.

The upper layer, called ProSys, is intended to design a system as a col-
lection of communicating subsystems executing concurrently and possibly dis-
tributed. In that layer, the subsystems are the components of the model and they
communicate together through asynchronous message passing between typed
message ports. This communication style is suitable at this level of granularity,
since it allows transparent communication between subsystems independently
of their location on the same physical node or not.
In comparison, the lower layer, called ProSave, is used for detailed modelling of small parts of control functionality of subsystems allocated to a single node and interacting with the system environment through sensors and actuators. Building on the approved features for analysability of SaveCCM [30, 31], the “pipe and filter” paradigm as well as a restrictive semantics have been adopted for this layer. The only architectural entities are components as main abstraction for real-time tasks or control functions and connectors for special operations on the connection between the components.

The two layers are not independent but relate to each other, since ProSys component may be modelled out of ProSave components. For more detailed information about ProCom, the reader is referred to Paper C or [32].

3.3.4 Integration of Extra-Functional Properties in Component Models

As identified in Section 3.3.2, an important requirement in the development of embedded systems is the possibility to perform various types of analysis throughout the whole development starting from early analysis to more detailed analysis and verification later. To efficiently contribute to the development, these analysis techniques must be an intrinsic part of the approach and be tightly connected to the component model whenever this is possible. This implies that all the artefacts needed and produced by the analysis techniques should be easily accessible, refer to the appropriate entities of the component model and be managed in a systematic way to eventually automate the analysis. Additionally, the analysis results should be reused in a suitable way.

In this respect, this work proposes a way to specify, integrate and manage information within component models, and more specifically extra-functional properties. This work constitutes the second step towards conciliating analysis with the envisaged component-based approach, after having specified a component model with a restrictive semantics and limited number of architectural elements. The main purpose with this works is to provide an appropriate support allowing a closer integration of analysis with the component model, with the long-term vision of eventually enabling as many fully automated analysis and verification steps as possible.

To this end, this work started by looking at the huge diversity of extra-functional properties that can be defined and accordingly proposes a format for their specification in order to manage them in a systematic way. The main intention with this definition is to have an unambiguous and precise semantics both with respect to the meaning of the extra-functional property and to the
correct format for specifying value. Thus, through the concept of attribute, we define an extra-functional property as follows:

\[
\text{Attribute} = \langle \text{TypeIDentifier}, \text{Value}^+ \rangle \\
\text{Value} = \langle \text{Data}, \text{Metadata}, \text{ValidityCondition}^* \rangle 
\]

where:

- TypeIdentifier defines the extra-functional property in a unique and unambiguous way;
- Data contains the concrete value for the property;
- Metadata provides complementary information on data and allows to distinguish between them; and
- ValidityConditions describe the conditions under which the value is valid.

This definition implies that an attribute, i.e. an extra-functional property, can have multiple values identified by metadata or the conditions under which the values have been obtained, such as for instance some assumptions on the target platform specification. This particularity of our definition has emerged from the need to cover both the entire development process from early design up to synthesis and deployment phases and the relation with the target platform specification. More explanations concerning the terms used in this definition as well as discussion about multiple values and reusability of extra-functional properties can be found in Paper D.

In addition, techniques outside this definition are provided to ensure a systematic comprehension and utilisation of the attribute concept within a development context:

- Connection, through an extension of the metamodel, to the entities of the component model that can have attributes.

- Definition of an attribute registry to ensure the uniqueness of the attribute specification.

- Specification of composition and selection techniques.
3.3.5 Prototype Implementation

The main intention with this work is to evaluate from a practical angle the envisaged approach of merging component-based principles and embedded system development needs i.e. to establish the advantages, drawbacks and limitations of the approach. This requires an implementation of the complete development toolchain from design up to synthesis and deployment, including some analysis techniques. As the work on establishing the requirements for the elaboration of ProCom was still in its early phase, no analysis or synthesis techniques were available at the start of this research work. Instead, it has been decided to use the concepts, methods and techniques developed for SaveCCT [33] to develop a first prototype, since SaveCCT shares many similarities with the work presented in this thesis. In particular, it presents a simple use-case scenario of the envisaged approach in that sense that the use of component is restricted to the design only and the analysis is performed on system-scale.

![Diagram of the SaveCCT approach](image)

Figure 3.3: Overview of the SaveCCT approach.

Based on [8] and with respect to the SaveCCM reference manual [33] which defines the exchange format to be used between the tools, an integrated development environment, called Save-IDE, has been specified and developed. Compared to the majority of existing IDEs which focus mainly on programming aspects, the Save-IDE integrates the design, analysis, transformation, verification and synthesis activities as illustrated in Figure 3.3. These activities are supported by a set of dedicated tools. The complete description of the approach and the environment can be found in Paper E.
3.4 Methodology

Equally important as the proposed solutions to answer the research questions, is to adopt an appropriate research methodology helping guarantee the soundness and the reproducibility of the work. In this thesis, we followed a methodology adapted from the guidelines proposed by Shaw in [34] to perform good software engineering research.

This approach starts with the identification of a problem from the real world (Problem Identification), in our case the limitations of the current development methods for distributed embedded systems due to the increasing complexity of new embedded system functionalities. The problem is then transferred into a research setting to be investigated with the prospects of finding solutions to it. However, since real world problems are generally quite complex, the scope of the problem needs first to be restricted to be manageable within a research context (Problem Setting). This limitation made us focus on a particular aspect of the real problem by formulating the research problem that will be addressed within the work (Problem Formulation), and then by stating Working Assumptions and Research Questions, which together set a frame for the work.

Similarly to passing from a real world problem to a research problem, breaking down the research problem into a set of research questions narrows down even further the problem to investigate and helps on focusing on particular aspects of the research problem. In that sense, the working assumptions provide a starting point to the work whereas the research questions correspond more to the specification of the angle of attack chosen to investigate the research problem.

Once the problem to address is clearly defined, the research work starts with the study of related theories, methods, approaches, techniques or solutions that have already been performed on the topic (Background Theories). With the knowledge of the existing state-of-the-art and the questions to answer, some solutions can be devised (Solutions). Formulating solutions is not a straightforward process but an iterative one, in which preliminary ideas are formulated, worked out, refined or even sometimes left aside. When the ideas are mature enough, they must be evaluated and validated to check whether they really answer the research question in a suitable way (Validation). If this step fails, the proposed solutions need again to be revisited, refined, improved or thrown away. In that sense, this is an iterative trial and error process, in which analysing the causes of the erroneous solutions might provide useful inputs to find new, better or simply working solutions.

After the validation step is satisfied, the applicability of the proposed solutions to solve the real-world problem can be evaluated (Evaluation). An
The work presented in this thesis is concerned with the problem identification, problem setting and research work steps. The validation and evaluation steps remains as future work. Each research questions can be answered in different ways and in applying different approaches, thus we describe below the methodology that has been used in the research work described in the previous sections.

The process to answer the first research question started by studying both the needs in the development process of distributed embedded systems and the current state-of-the-art of component-based software engineering focusing on existing component models, in particular SaveCCM [8]. This study was based on literature surveys and discussions with domain experts of vehicular and automation domains. Based on these findings, requirements for the component model were extracted and served as foundations in the elaboration of ProCom, which addresses some of the limitations of SaveCCM.

As for the work concerned which research question 2, it also started with a literature surveys on extra-functional properties and their management and the identification of a few properties of interest in the development process. Then
we have tried to relate their management to their utilisation within the development process. The methodology followed here was iterative and started with the development of a prototype implementing some preliminary ideas to get a better understanding of their integrations and contributions in the development process. This preliminary solution has then been refined into the attribute framework presented in Paper D.

The last research question was concerned with the feasibility of combining a component-based approach with formal early analysis. We proceeded by construction and realisation of an integrated development environment that provided us useful lessons learned.
Chapter 4

Related Work

In this chapter, we relate the contributions presented in this thesis, namely a new component model for distributed embedded systems, a framework to manage extra-functional properties and an integrated development environment, to similar relevant approaches.

4.1 Component Models

A broad range of component models exists nowadays, either general purpose or dedicated component models, as compiled in various classifications (as in [4] or [35] for instance). However few component models actually target the development of embedded systems and most of them focus on a specific domain only. Using the component models detailed in Paper A as a basis, this section goes back over the component models targeting embedded systems and compares them with the component model proposed in this thesis.

In the automotive domain, the AUTOSAR (AUTomotive Open System ARchitecture) consortium [1] is the first large-scaled initiative to gather manufacturers, suppliers and tool developers from the automotive field to establish an open and standardised software architecture for the automotive domain enabling component-based software design modelling. Through this common standard, the vision of AUTOSAR is to facilitate the exchange of solutions (including software components) between different vehicle platforms and subsystem manufacturers as well as between vehicle product lines. In that sense, AUTOSAR targets the upper part of the granularity scale of the proposed
conceptual component model. Similar to our approach, AUTOSAR relies upon the use of a component-based software design model. However the two approaches have principal differences. In particular, AUTOSAR component model proposes both pipe and filter and client-server paradigms communicating transparently across the architecture through the use of standardised interfaces. Although targeting development of applications for the automotive domain, AUTOSAR in its current version lacks support to express and analyse extra-functional properties in particular timing properties as for instance worst-case execution time or end-to-end deadline. An upcoming release AUTOSAR 4.0, done in cooperation with the TIMMO project [36] and EAST-ADL [37], intends to tackle this lack by an extension of the current metamodel. In particular, the TIMMO project intends to propose a standardised infrastructure to manage timing properties and enable their analysis at all abstraction levels from early design to deployment.

A second initiative that shows the growing interest from the automotive domain in component-based software development comes from Bosch with BlueArX [6, 38]. Also based on a design-time component model, BlueArX differentiates itself from AUTOSAR in supporting timing and other non-functional requirements as well as in focusing on complete development process for single ECUs. To this respect, BlueArX is relatively close to the objectives and contributions presented in this thesis in particular with regards to the lower layer of the component model (ProSave). However differences exist. First, through the ProSys layer of the component model, ProCom intends to support also the development of embedded software systems distributed across several ECUs. Another difference lays in the proposed support to integrate analysis. Whereas extra-functional properties can be associated with any entities of the ProCom component model (components, ports, services, connections or component instances) through the attribute framework extension, BlueArX on the other hand endows components with an additional analytical interface to perform analysis either at system- or component-scale. In a recent work [7], BlueArX has been extended to support the analysis of timing properties in relation to operational mode, a feature which is not supported yet within ProCom.

Developed in a close cooperation between Arcticus Systems AB and Mälardalen University, the Rubus Component Technology [9] is another example of an industrial use of component-based approach in the vehicular domain. Similarly to ProCom, the RUBUS component model focuses on expressivity and analysability through a restrictive component model. However, the Rubus component model allows the specification of timing properties only and is not primarily concerned with reuse.
The contributions found in this thesis are largely inspired by previous work done at Mälardalen University on the elaboration of a component model for vehicular domain. SaveCCM [33] is a design-time component model consisting of a few design entities with a restrictive “Read-Execute-Write” execution semantics and communicating through a “pipe & filter” paradigm in which the control- and data-flows are distinctly separated. Having such a restrictive semantics, it enables formal validation and verification of the system already in early phase of the development process, prior any implementation as well as automated part of the transformations into an executable system as explained in [39]. ProCom is built on the knowledge and experiment gained from the development of SaveCCM and tries to alleviate some of the restrictions and drawbacks of SaveCCM in particular in strengthening the concept of components, considering distribution and handling functional and extra-functional properties in a more systematic way. Whereas the ProSave layer is to a large extent directly inspired from SaveCCM, the upper layer (ProSys) aims at addressing the distribution of subsystems, which was not addressed within SaveCCM.

In the field of consumer electronics, Philips has developed and successfully used the Koala component model [10] for the production of various consumer electronic product families (TV, DVD, etc.). In comparison to the aforementioned initiatives, Koala is less oriented towards safety-critical applications than what exists in the automotive domain for example. However, as Koala still targets severely constrained embedded systems, it pays a special attention to static resource usage, such as static memory for instance, but it lacks support for managing other extra-functional properties. The dependencies between properties are handled through diversity spreadsheet, which is a mechanism outside the component. Koala has served as input in the Robocop [40] project done in collaboration between Philips and Eindhoven Technical University. Similarly to ProCom, Robocop considers components as a collection of models covering the different aspects of the development process. Models are also used to manage extra-functional properties as for instance the resource model, which describes the resource consumption of components in terms of mathematical cost functions, or the behavioural model, which specifies the sequence in which the operations of the component must be invoked. Additional models can be created.

Pecos [41] is a joined project between ABB Corporate Research and Bern University. Its goal is to provide an environment that supports specification, composition, configuration checking and deployment for a specific type of reactive embedded systems (field devices) built from software components. Contrary to ProCom for which the components of each layer have their own
execution semantics, i.e. ProSys components are active whereas ProSave components are passive, the two types are put together in Pecos. Also, since components in Pecos have only data ports, there is a need for an additional type of component, called event component, which activation is triggered by the arrival of an event. With regards to extra-functional properties, Pecos enables the specification in a name-value pair format in order to investigate the prediction of the timing and memory usage of embedded systems. However, this specification is limited to name-value pairs in difference to the possibility offered to specify extra-functional properties in ProCom.

Pin [42], a component model developed at Carnegie Mellon Software Engineering Institute (SEI), serves as basis for the prediction-enabled component technologies (PECTs) which aims at attaining predictability of run-time properties such as performance, safety and security. Alike our approach, PECT stresses the importance of providing suitable quality prediction based on analysis theories. However the methods to integrate analysis differ. Whereas ProCom relies on an external attribute framework as means to handle functional and extra-functional properties resulting from different analysis techniques, PECT is centered around a reasoning framework consisting of analytical interfaces used to specify specific properties, and corresponding analysis theories to enable the prediction of these properties. Also in comparison to ProCom, Pin is a flat component model which does not support distribution.

4.2 Alternative Approaches

This section correlates our work with other approaches that are not primarily concerned with the principles and methods advocated in CBSE but are still intended to support the development of distributed embedded systems.

In the automation domain, the standards IEC-61131 [21] and its successor IEC-61499 [43] proposed by the International Electrotechnical Commission are well established technologies for the design of Programmable Logic Controllers. Whereas IEC-61131 allows to graphically compose systems out of function blocks, IEC-61499 has been developed to enforce encapsulation and provide a support for distribution. From a design perspective, ProCom shares some similarities with these graphical languages, in particular the encapsulated entities communicating with a “pipe & filter” paradigm with explicit separation between data- and control-flow, and the distribution support. However the semantics associated with the function blocks are weaker compared to the ProCom components, and the standards lack support for specifying and managing
extra-functional properties and their analysis. This holds back the possibility for formal analysis of the systems under development, which is one of the major objectives this thesis aims at.

In the automotive domain, alike ProCom, EAST-ADL (Electronic Architecture and Software Technology – Architecture Description Language) [37] aims at providing a support for the complete development of distributed embedded systems by taking into consideration the hardware, software and environment development assets. Although both approaches share similar objectives, they differ in the way those objectives are approached. Whereas ProCom emphasizes components as assets for capturing development information thus aiming at reusability, EAST-ADL focuses on architecture description to structure it. In EAST-ADL information is structured into five abstraction levels, which describe the functionalities from several standpoints. Each entity of a level realizes the entities of the higher abstraction levels. ProCom covers three of these levels (analysis level, design level and implementation level), and leaves out the electronic feature design (vehicle level) and the support for the deployment of the final binary (operational level). Similarly to ProCom, EAST-ADL also supports modelling of non-structural aspects such as behavioural description but covers in addition validation and verification activities as well as management of requirements. EAST-ADL was originally developed as an EAST-EEA ITEA project involving car manufacturers and suppliers and now it is refined as a part of ATESSST project to be aligned with the major standardization efforts existing in the automotive and real-time domains (AUTOSAR, MARTE, and SysML).

The Architecture and Analysis Description Language (AADL) [44], formerly known as Avionics Architecture Description Language, is a standardization effort led by the Society of Automotive Engineers (SAE) to provide support for the development of real-time and safety-critical embedded systems for aerospace, avionics, robotic and automotive domains. Consequently, AADL stresses the importance of analysis to meet the particular constraints and requirements of the envisaged target domains. It provides a formal hierarchical description of the systems including properties to support the use of various formal analysis techniques related to timing, resources, safety and reliability with the aim of validating, verifying and performing tradeoff analysis of the system. Properties are defined as a triple (Name, Type, Value) that can be attached to different entities and can have specific instance values. To this respect, AADL is comparable to ProCom and its attribute framework. However, in comparison to ProCom, AADL is “only” a description language and does not provide links to design and implementation technologies. In that sense, it
decomposes the system in a top-down manner specifying entities and how they interact and are integrated together without providing any implementation details. Thus AADL is not primarily concerned with reusability issues. On the other hand, AADL includes some features that could be interesting to take into consideration in the further development of ProCom such as the specification of execution platforms and operational modes.

Other approaches apply Model-Driven Engineering (MDE) techniques that allow to automate the development process in relying on models as primary development artefacts, hence abstracting away from implementation concerns. These models are intended to serve as input to automatically derive implementation, documentation, test cases, and much more. Although not limited to UML-based models, the attractiveness of these approaches has increased since the introduction of UML 2.0 [45] and various UML-profiles such as SysML [46] and MARTE [47]. In particular MARTE, the successor of the scheduling, performance and timing (SPT) profile, defines a set of basic concepts for model-driven development of real-time embedded systems. In that sense, MARTE is closely related to the work presented in this thesis, especially with the specification of extra-functional properties including time and resources and the intention to support various types of model-based analysis such as schedulability and performance. In addition, through the General Component Model sub-profile, MARTE proposes support for CBSE. However contrasting to our work, MARTE does not focus on implementation and reuse.

4.3 Integrated Development Environment

Integrated Development Environments (IDEs) are not new. They traditionally provide dedicated support for developing applications in various programming languages such as Pascal, C/C++, Java, PhP among many others. In these environments, the main focus is oriented towards the implementation phase of the development process, which means that typically source code editors (with syntax highlighting, auto-completion, bracket matching, etc.), compiler and/or interpreters, and debuggers are supplied to the developers. For object-oriented software development, class browsers, object inspectors, and class hierarchy diagrams, are also integrated. The most common representatives of these IDEs include Delphi [48], Eclipse [49], and Microsoft Visual Studio [50].

As for CBSE, the environments are generally tightly centered around a component model, and focus on specific development phases (implementation) and domain. Some examples of such environment are Palladio Component Model tool [51], Koala Development Tools, Netbeans [52] for EJB and Jav-
aBeans.

However for the development of safety-critical real-time embedded systems, environments providing more verification and simulation capabilities are often used instead — either a UML-based environment or a dedicated environment. In those environments, code generation is a rather common feature, which allows to automatically derive accurate implementation from models. In some cases, as for BridgePoint [53], the generated implementation can be executed directly to simulate the behaviour of the system.

UML-based environments propose to develop a system in starting by its design following UML or a UML-profile. Typically, those environments cover design, code generation, execution, tests, and simulation. Despite the recent initiatives of SPT, SysML and MARTE to incorporate extra-functional properties into UML, few tools actually support those new standards and when this support exists, it still lacks formal grounds. As a result no automatic verification is available in those environments. However, through combining UML class diagrams and UML behaviour diagrams, the Fujaba Tool Suite [54] manages to enable formal system design that can be used to generate Java source code. Rational Rose Technical Developer [55], Rhapsody [56] and BridgePoint are some examples of environments belonging to this category.

On the other hand, dedicated integrated development environments are centered around a dedicated modelling language. Simulink [57], from MathWorks, is the leader environment to model, simulate, implement and analyze dynamic and embedded systems. It is mainly used in control theory and digital signal processing for designing the applications together with modelling its environment. Once the system is designed out of block diagrams (very similar to components), the system can be synthesized into executable code through a connection to the Real-Time Workshop tool also developed by MathWorks. A repository support is provided on the form of building block libraries, from which building blocks are picked and customized to fit the needs of the new design. Simulink is integrated with Matlab, hence allowing algorithm development, data visualization, data analysis, and numeric computation. Other major dedicated environment is SCADE [58], which proposes an environment to produce mission and safety-critical systems mainly for aerospace, defence and automotive domains. SCADE is endowed with the following features: graphical and textual editors, simulator, formal proof (Design Verifier), code generators, model test coverage and can be connected to Simulink, DOORS, Altia, UML/SysML, etc.

Save-IDE, described in Paper E, also belongs to this category. In comparison to the other environments, Fujaba Tool Suite excepted, Save-IDE provides
an environment allowing formal modelling of a system fully compliant with the SaveCCM semantics, hence enabling formal verification of the behaviour of the system with respect to time, safety and reachability properties. However, in order to benefit from the large variety of existing tools for UML, Save-IDE through its SaveUML extension [59] allows transforming a SaveCCM design into a UML-profile and vice-versa.
Chapter 5

Conclusions and Future Work

We have described in this thesis a possible approach to merge component-based software engineering principles with the specifics of distributed embedded systems with the aim of providing solutions towards an efficient development environment. This approach is based on a dedicated component model tailored to fit the embedded system development needs. In particular it provides a restricted semantics to facilitate the analysability of the system being designed and a dedicated (extra-) functional properties framework to ease the integration and management of analysis techniques and their outputs.

This chapter concludes the thesis by reviewing and discussing its contributions with regards to the research questions stated in Chapter 3, and by providing directions for future work.

5.1 Discussions

Clearly, the objective of proposing integrated solutions to develop distributed embedded systems in a predictable and efficient way while following the CBSE principles is ambitious. It can be addressed in many different ways and requires many fragmentary results which need to tightly fit together. This objective is not attained entirely through the contributions presented within this thesis since the work is not completed yet. However, we have provided basic foundations and directions, which hopefully contribute to move closer to its realisation.
The main piece of remaining work concerns the validation and evaluation of the proposed methods, in particular with respect to the envisaged component-based approach and its underlying component model and extra-functional property support. Indeed, no validation or evaluation in an industrial context has been performed yet and this constitutes an important part of future work that remains to be done. As a consequence, the answers to the research questions 1 and 2 provided below correspond more to initial findings on the subject than fully accepted results corroborated through the development of suitable applications or even industrial case-studies. The remainder of this section provides answers to the research questions introduced in Section 3.2 and discussions on their relevance.

**Research Question 1:**

What are the suitable characteristics of a component model to efficiently support software design of distributed embedded systems?

Based on an analysis of the component model classification framework and an evaluation of the requirements for embedded system development, a number of characteristics that seem suitable for component-based embedded system development and its associated component model have been identified and detailed in Paper B and integrated in ProCom (Paper C). As our answer to the Research Question 1, yet to be confirmed by experiments or case-studies, a component model should support:

- Different abstraction levels (i.e. the coexistence of components in an early design phase and fully realised components).
- The different concerns that exist at different granularity levels (i.e. an high-level view of loosely coupled complex subsystems together with a low-level view of small non-distributed functionalities similar to control loops).
- Platform awareness while still being platform independent.
- Various analysis techniques.

In addition, as identified in [30] for the development of embedded control software, the component model semantics should also be limited and restrictive to support important extra-functional properties such as timing, safety or reliability. With regards to efficiency of software development, this implies finding the appropriate tradeoff between flexibility on one hand and analysability and predictability on the other hand. We approached this problem by alleviating some
of the restrictions present in SaveCCM — in particular for the ProSys level which requires more flexibility than ProSave since it deals with distributed active subsystems executing concurrently — while reinforcing the concept of components as a unified notion throughout the development process. In spite of this, ProCom provides a semantics precise enough to be formally expressed through timed finite state machines as demonstrated in [60]. Similar to what has been done in SaveCCM, this should permit an automated integration of formal analysis tools, improving the development process performance.

The strong coupling between target platform specification and software implementation is an important challenge which requires to be addressed in a suitable way since the correctness of analysis results and values of extra-functional properties strongly depend upon the target platform specification and the deployment configuration. Postponing the access to this information to a late development stage could result in incorrect design and implementation of the system to be executed, leading to an eventual costly redesign and re-implementation of the erroneous parts of the system. Yet, breaking the hardware abstraction and making the target specification part of the component model is not a suitable solution since this would make all components platform dependant and hinder their reusability breaking then one fundament of CBSE. An appropriate solution lays probably in between those two extreme solutions.

**Research Question 2:**

*How to provide efficient integration support for management of functional and extra-functional properties within a component model?*

Answering this question corresponds to finding an appropriate way to specify, integrate and handle functional and extra-functional properties in a component model in a systematic way. Thus, we addressed this question through the approach briefly described in Section 3.3.4 and detailed in Paper D.

This approach combines a model for specifying extra-functional properties with techniques outside this specification, such as a property registry and property selection, to ensure the correctness of their utilisation in the current development context. Remarkably, a distinctive feature of our model lays in its ability to handle the specification of multiple values for a property, where each value is identified through the provision of suitable metadata and/or the context under which the value has been obtained. This approach can also be used to integrate the specification of functional properties without hampering the utilisation of interfaces. In this context, functional properties do not refer
to interface specification of the operations handled by the components, but to the modelling of the behaviour of the components in a format suitable for analysis techniques such as timed automata model. By this means, our intention is to increase the analysability and predictability of component-based embedded systems, and enabling a seamless and uniform integration of existing analysis and predictions theories into component models.

However this solution introduces complexity in the design process in several ways. In addition to the possibility to have multiple values assessed at different point of time or by different techniques, it also envisions delegating the declarations of needed properties to, for example, the developers of the analysis techniques who know best the types of information they need as input and that they produce as outputs. In the end this could result in an explosion of property definitions in the registry. A possible solution would be to rely on a standardized catalogue of properties similarly to what exists for units (SI), date and time representation (ISO 8601) or the standard for evaluation of software quality (ISO 9126).

Our approach to integrate extra-functional properties in component models reveals a lot of information concerning the details of the implementation of the components. Although this is not a major issue for in-house development, it naturally becomes more problematic for its utilisation in the development of systems or components for which the implementation details must remain hidden such as COTS components since all the models that have served for analysis are packaged together with the components. A solution could be to provide mechanisms to identify and automatically remove confidential information when components are distributed to third parties.

Research Question 3:

How to build an integrated development environment encapsulating suitable models and technologies to efficiently support component-based development of software for embedded systems?

Based on [8] and with respect to the SaveCCM reference manual [33] which defines the exchange format to be used between the tools, an integrated development environment, called Save-IDE, has been specified and developed. This environment has been used internally by the members involved in its realisation but also externally by students outside the projects to develop diverse small applications. In [61], a comparison between Save-IDE and a professional tool enhanced with a profile for SaveCCM has been performed. This experiment is performed on a small group of students concerns only the modelling aspect of
the environment. Yet the students’ feedback show some indications that a dedicated design environment is more efficient than a general-purpose environment customized to fit a particular need. So as a part of the answer to the research question, a first important feature is the presence of dedicated modelling editors. The environment has also been used in [62] and in [39], in which an industrial control system and a simple truck application have been realized respectively. Those two examples show the feasibility of the integrated approach. In particular, they highlight the possibilities of tightly interconnecting design and formal analysis tools, which enable formal analysis of the on-going design already in an early design phase.

From the development and internal use of this environment, several conclusions have been drawn, leading to some areas of improvement for the environment and some of them as served as basis in the on-going work to develop an integrated development environment supporting the component-based approach presented in this thesis. These conclusions are other parts of our answer to this research question.

The first conclusion is that components must be the main unit of development, similar to the concept of packages in object-oriented programming, and must be manipulated as such. In that view, a component is the collection of its data and files such as architectural model, behavioural models, source code, tests, documentation, etc., which must be kept consistent. This should enable component versioning, foster bottom-up development with possibly reuse, and ease the distinction between component types and instances, which was one of the problem faced in Save-IDE. Indeed in Save-IDE, components are design entities only, and are created during the design of the system through the architectural editor. One problem with this approach is that it is difficult to determine when the design of the component is completed and must not be changed any longer. The possibility to copy components in the design adds to the problem even further since it implies that component types and instances are mixed together. This means that an instance of the component can be modified independently of its component type and consequently, ensuring consistencies of a component type with its instances and implementation requires numerous checking.

Another conclusion is that information concerning the platform design must also be highly interconnected with the software design so that parameters from the target platform specification that influences the software design are available as soon as they are specified and vice versa. This could enable the integration of analysis tools which requires knowledge on the platform to produce accurate results.
Finally, in the current approach supported by Save-IDE, the transformation of the design model into an execution model allowing synthesis and optimisation steps is performed at the end of the process only, after the design has been verified and validated. Yet, the validation and verification are performed at a high-level of abstraction without connection to the component implementation used in the synthesis and without any specific information regarding the target platform. It is assumed that the implementation does not break the behaviour formally modelled. This can have some negative effects on the efficiency of the approach when the fully implemented system does not meet its timing requirements or the timing requirements are not feasible. The development process might then start over at the design step with the re-design and re-implementations of the erroneous parts. As a consequence, the validation and verification steps must be carried out again. Furthermore some analysis techniques, such as schedulability, cannot be performed on a high-level of abstraction. Some potential solutions that need to be further investigated are to connect implementation with analysis or generating implementation from the models used by the analysis techniques. Also, synthesis must be viewed as more complex than a single-step operation performed at the end of the development process. It requires many analysis, tests and optimisations that are closely related to the design, implementation and various extra-functional properties such as timing or resource usage, and must therefore be also tightly connected with them.

5.2 Future Work

An important part of the work that currently remains concerns the evaluation and validation of the proposed methods. To complete this work, we are currently building the PROGRESS IDE, an integrated development environment centered around the notion of component as main unit of development and supporting the requirements of the proposed component-based approach. In particular, this IDE is intended to support the co-existence of fully implemented components with components in early design phase, and emphasize reuse. We envisage to use this integrated development environment to conduct experiments and case-studies addressing the development of embedded systems, primarily with regards to the vehicular domain. Later, we also plan to evaluate the applicability of the proposed methods for other domains such as automation and telecommunication.
In addition, since specifying and building an efficient and predictable software development framework for embedded systems requires many results tightly interconnected to each other, the work presented in the thesis can continue in several directions. Some of them are:

- Investigating target platform specification together with mechanisms to connect information to the software design when appropriate and reciprocally, relate design information to the target platform design, hence establishing suitable relationships between software and hardware designs.

- Improving the attribute framework by refining the validity conditions and the automated selection of attributes, and supporting the migration of information between component instances and types.

- Further elaborating ProCom for handling sensors and actuators, and at the ProSys level, supporting additional communication paradigms, such as synchronous communication.

- Integrating analysis techniques and their underlying models and more specifically REMES, a resource model for embedded systems that can be used for early analysis of timing and resource usage.

- Considering synthesis as a multi-step activity and investigating its relationships between software design and target platform design.
Bibliography


[51] Karlsruhe Institute of Technology and Research Center for Information Technology. Palladio Component Model Tool. 
http://sdqweb.ipd.uka.de/wiki/Palladio_Component_Model.

[52] SUN MICROSYSTEMS, INC. Netbeans. 


http://wwwcs.uni-paderborn.de/cs/fujaba/.


[56] IBM Rational. Rhapsody. 

[57] Simulink, MathWorks. 

[58] ESTEREL Technologies. SCADE Suite. 

