Support for Hardware Devices in Component Models for Embedded Systems

Luka Lednicki
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
luka.lednicki@fer.hr

Abstract — **With the decreasing costs of electronic parts for embedded systems, complexity of their software has drastically increased. A possible solution for handling this high complexity is component-based development, a branch of software engineering that builds complex software systems out of encapsulated units of software named software components. Component-based approach has proven beneficial in enterprise systems and desktop domains. However, embedded system domain introduces some domain-specific problems (e.g. satisfying safety-criticality, real-time requirements and interaction with environment). Therefore, if we want to use the component-based approach in embedded systems we must address these problems. In this paper we present an overview of how interaction with environment impacts the use of component-based approach for embedded systems. We present different ways in which component models can enable us to manage hardware devices and provide examples from existing component models. We also present our research plan that addresses the need to improve how component models enable managing hardware devices.**

Component-based software enginveering; hardware components; hardware devices; non-functional properties; analysis

I. Introduction

As modern embedded systems grow in complexity component-based development is an increasingly attractive approach utilized to make the development of such systems simpler and less error prone [3]. However, in embedded system domain component-based approach to software development is seldom used in practice, and is mostly explored in component models used in research context.

Amongst other things, one of the problems that component-based development for embedded systems must address is interaction of a software system with the environment, the physical world that the system is embedded into [5]. This interaction is done using *hardware* devices, such as sensors and actuators. A simple example of an embedded system is a temperature regulation system, which keeps an constant temperature in a room, cooling or heating the air in it. Such a system must have at least one temperature sensor, and two actuators: one for starting the heating process and one for starting the cooling process. So, even if complexity of software in such a system is very low, its behavior is highly dependent on the communication with hardware devices, and behavior of devices themselves.

Communication between software and hardware devices can be as simple as writing value to a hardware pin or port of the device that the system is deployed on, or as complex as invocation of a service on a remote device. In all cases, this interaction with the environment implies dependencies of software components on the hardware or middleware used to communicate with the environment. Same environment, and combination of hardware and middleware also affect the behavior of an embedded system. As reusability and analyzability of software components and component based systems highly rely on such dependencies and effects on behavior, failure to adequately express them can hinder the use of component-based approach in the embedded system domain.

Section II of this paper provides a background on component-based development for embedded systems. In Section III we discuss different ways in which hardware devices can impact component models. Section IV presents four levels of support for hardware devices that component models can provide. In Section V we give an overview on some of existing component models, showing their level of support for hardware devices. Our plans for future research are given in Section VI. Section VII concludes the paper.

II. Component-based Development for Embedded Systems

Component-based development [4] is a software engineering approach in which software systems are built by composing them out of preexisting and reusable units of software, software components. However, these components are more than just segments of software code. In many cases software components are packages of containing software code, different models that describe behavior of the component, collection of both functional and non-functional properties or attributes and different documentation files that describe a component. In this way, beside just the benefit of reusing components and reducing efforts needed for system development by composing systems out of components, we are able do different types of analysis of system properties and behavior in early stages of system development, before the actual system is complete and deployed [6].

Non-functional properties like worst-case response time of some functionality, or memory consumption, are often as important when developing software for embedded systems as is their main functionality. Because of their small size, limited power sources and limitations

on cost embedded systems generally have poor processing power and memory resources compared to standard desktop computers. Also, their functionality is often time (real-time systems) and safety critical (e.g. vehicular electronic control units). Ability to conduct analysis in early phases of system development, and predict system behavior and properties (both functional and non-functional) can greatly improve development process of such systems. For this reasons there are many component model that aim embedded system domain, e.g. SaveCCM [12], Rubus [10], COMDES-II [11], ProCom [15], AUTOSAR [1], Koala [14], etc.

As already mentioned, all possible component-based development for embedded systems is still not fully exploited. Component models used in industry do not provide all the potential benefits of component-based development, and are mostly used just for system modeling. Also, most of the component models used in industry provide support for hardware device which does not promote reuse of hardware-dependent components and limits our ability of system analysis. On the other hand, component models for embedded systems currently used in research most often focus on providing support for handling only pure software components and rarely try to provide comprehensive approach for dealing with components dependent on external devices, which is essential for real-world use.

III. Effects of Hardware devices on Software Component Models

Dependencies on hardware devices can affect component models on many different levels. As main functionality of embedded systems is that they interact with their environment using hardware components, to be able to fully utilize component-based approach in embedded systems we must first identify how these dependencies affect component models.

From the architectural point of view, such dependencies have to be clearly stated and presented for developers to be able to see and manage them.

In the deployment phase, software components and subsystems are allocated to the underlying hardware that will support them. In this phase, there must be an ability to see dependencies on hardware devices and ensure that the hardware targeted for deployment can satisfy these dependencies.

During analysis phase, effects of the external devices on behavior of software components must be taken into consideration.

Finally, during synthesis phase we generate executable code using system models. In this phase we must take care that the code generated for software components reflects the platform's specifics of communicating with external devices.

As reuse of once developed components is one of key concepts of CBD, we also need to make sure that components that are dependent on hardware can be deployed on different platforms that handle hardware devices in different manner.

IV. Approaches on Integrating Hardware devices

We have identified four main levels of support for hardware devices that component models can provide. These levels are depicted in Figure 1.

A. Externalized, outside of component model

Some component models avoid providing any support for stating dependencies or communicating with hardware. Interaction of software components with environment is forbidden. Instead, it is supposed that this communication is handled by software outside of component model framework, and then in some way

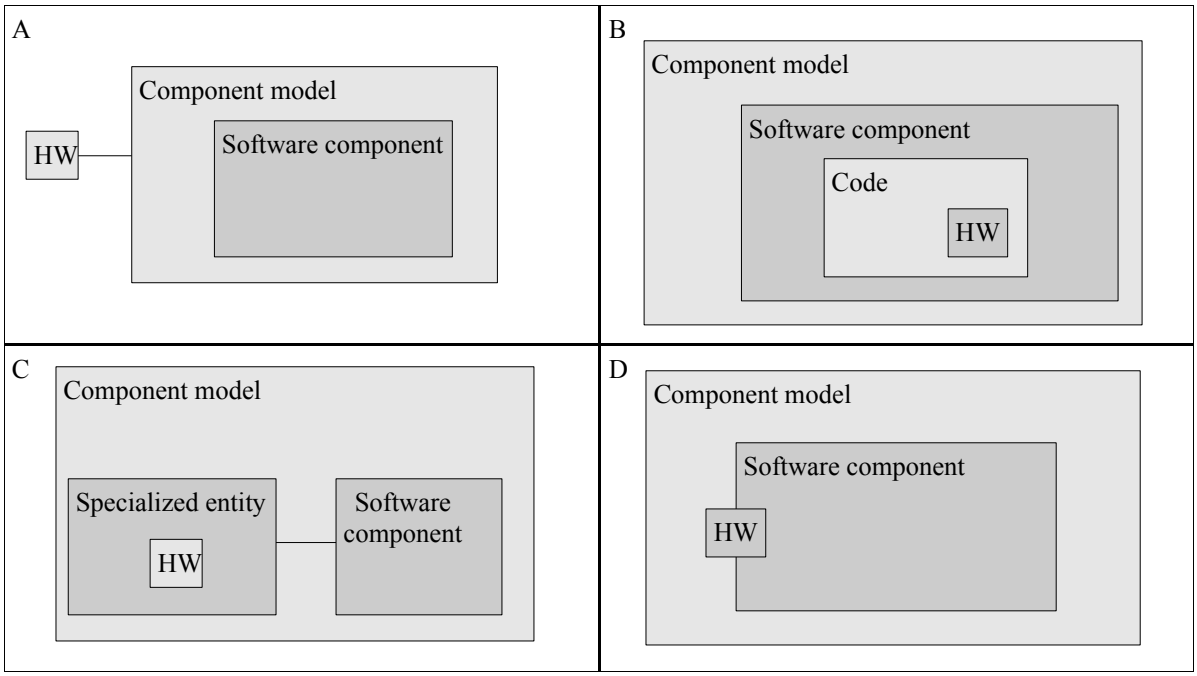

Figure 1. Different levels of support for hardware devices in component models: A – externalized, outside component model; B - Implicitly, on code level; C – explicitly, using specialized entities; D – explicitly, encapsulated in software components.

presented as inputs or outputs to the framework and system composed out of components.

Although this is a valid approach, it is obviously inadequate for use in embedded systems as main task of embedded systems is communication with their environment. By not having a way for expressing interaction with environment, these types of component models can make system development more cumbersome and less suitable for analysis.

B. Implicitly, on code level

It is common in component models to handle interaction with environment as any other code inside software components. In this case software components communicate with hardware by direct method calls to underlying platform or operating system. These method calls are interleaved with the rest of the code, and hard-coded inside component.

Such treatment of communication with environment limits our ability to fully utilize advantages of component-based development in embedded systems. First of all, it stops us from using a component as black-box as it interacts with its environment in a way that is not visible from it's outside interface. Along with that, having platform-specific communication hard-coded inside component's code can greatly hinder reuse of the component, as it can only be reused on the same platform with exactly same configuration. This also prevents us from checking validity of deployment because we cannot determine if the platform we are deploying on is adequate for our system. Our ability to analyze such systems is also reduced, because we are unable to take into account non-functional properties and behavior of hardware components.

C. Explicitly, using specialized entities

Another way of handling hardware dependencies and interaction with environment is introducing special entities in component model that will encapsulate them. These entities are able to communicate with normal software components, but have syntax and semantics that differ from software components.

This approach is appropriate for component-based development for embedded systems as it enables us to explicitly state the dependencies of systems on hardware components and how software components interact with the environment. Reuse of software components is simple, as there is a clear boundary between software and hardware. It also enables us to include hardware components in system analysis.

One negative aspect of this approach is how it effects hierarchical composition of components. It is a common case in component models that they enable creation of *composite* components, i.e. software components that are composed of other software components (instead of defining their functionality by code). Composite components then act as normal software components, respecting same interface syntax and semantics. If we encapsulate hardware components in separate entities we have no way of exposing hardware devices in composite components, as hardware dependencies are not part of standard component interface.

D. Explicitly, encapsulated in software components

In some component models interaction with hardware is inclosed in software components, but exposed through component's interface. In this case we are able to describe communication with hardware components using same syntax and semantics as communication between normal software components.

Similarly to approach in subsection IV.C., this approach allows us to fully utilize component-based approach for developing software for embedded systems. Additionally, by having the ability to state interaction with hardware components as a part of software component interface we are able to include hardware components in a hierarchical software component model.

V. Examples of Hardware Component Integration

In this section we will give an overview of how some of the component models currently used in research or industry deal with dependencies on hardware components and how the communication between hardware and software components is treated. This investigation was done as part of research for survey on component models currently in use [9]. Although our research covered a number of component models for embedded systems, most of them did not provide any documentation or other information on how interactions with hardware components is treated. As a conclusion, we can argue that their support for hardware devices is either implicit (encapsulated in code) or externalized (outside of component model).

A. SaveCCM

SaveCCM [12] is a component model developed at Mälardalen university for purpose of research on component-based development for vehicular and safety-critical embedded systems. In SaveCCM software components are not allowed to directly communicate with hardware devices (hardware devices are externalized). Instead, communication with them takes place outside of the component model. Data that is the subject of the communication is presented as input or output values to the component-based system.

B. Rubus

Rubus component model [2], [10] was created by Articus Systems for development of dependable embedded real-time systems. Architectural elements of Rubus are software items, which can be either basic software circuits or assemblies or composites of other software items. Behavior of a software circuit is defined by a C-language entry function. There are no special architectural elements that model external devices such as sensors and actuators. Instead, they are modeled by basic software circuits (support for hardware devices is implicit, on code level). Sensors are represented by software circuits that have no input data ports and at least one input trigger port, while actuators are modeled by software circuits that have no output data ports. Platform and device dependent information or behavior are hard-coded in the software circuit's C entry function.

C. COMDES-II

COMDES-II [11] is a component-based software framework aimed for efficient development of reliable distributed embedded control systems with hard real-time requirements. COMDES-II defines a two-layer component model, having the "upper" layer specify the behavior of a systems using active software artifacts called actors, while the "lower" layer defines the behavior of the actors using function block instances. Interaction of actors with the environment is encapsulated in input and output signal drivers (hardware devices are supported explicitly, using separate entities). Drivers can be classified as either communication drivers (used to sense or actuate signals on a network), or physical drivers (used for sensing or actuating physical signals).

D. AUTOSAR

AUTOSAR [1], [8] is a new standardized architecture created by a partnership of a number of automotive manufacturers and suppliers with a goal to manage increasing complexity of vehicular embedded systems, enable detection of errors in early design phase and improve flexibility, scalability, quality and reliability of such systems. To achieve this, AUTOSAR applies component based approach for developing embedded systems. In AUTOSAR, underlying hardware that the system is deployed on is abstracted away by the AUTOSAR Run-time Environment that provides a platform-independent framework for the application layer. As a consequence, AUTOSAR applications can only be deployed on a hardware device only if there is an existing AUTOSAR Run-time environment for this specific device. In the application layer dependencies on specific hardware is encapsulated in special type of components, sensor and actuator components. These components are dependent on a specific sensor or actuator, but are independent of the hardware device that the application is deployed on. The fact that there is no possibility of hierarchical nesting of components software components and sensor and actuator components still act as separate entities, so we can not say AUTOSAR fully supports explicit encapsulation of hardware in software components. Support for hardware devices remains explicit, using separate entities.

VI. Proposed research

The goal of the research is to investigate to incorporating hardware devices into component models for embedded systems in such a way that and construct an approach that would significantly advance how these component models handle hardware devices.

Our hypothesis is that providing a way for explicitly stating dependencies of software components on hardware devices will enable reuse of such components on different platforms or different configurations of platforms. Also, by providing information about functional and non-functional properties of these devices, and providing a way to propagate these properties to component models, we will have a basis for better prediction of system behavior and the analysis of systems will be more accurate.

Some of the questions we will try to address are:

- What are the requirements for describing dependencies of software components on hardware devices in component models for embedded systems?
- How can we describe dependencies of software components on hardware devices and specify functional and non-functional properties of these devices?
- How can we improve reusability of software component that are dependent on hardware devices?
- How can we utilize information about functional and non-functional properties of hardware devices to enable more accurate analysis of systems?

Our research will be divided into three phases: defining requirements, hardware device description, integration into component models and evaluation. Next, we will describe each phase in more details.

A. Requirements

As the first part of our research we will need to investigate what are the requirements for describing dependencies of software components on external hardware devices. We can describe hardware devices on many different levels of abstraction and specify many different properties (both functional and non-functional) that characterize them. However, we must find which of these abstraction levels the is most adequate and which of the properties are required for successful integration of hardware devices into component models. We will especially have to take into account how hardware devices affect reuse and analysis of software components and component based systems.

B. Hardware device description

After we understand the requirements for describing hardware devices we must find a way to describe them. Because of the overall need for describing hardware in engineering community, there are already a number of hardware description languages (for example AADL [7] or SysML [13]) that are used to define hardware on different levels. Some of these languages can be easily incorporated into component models.

Taking into account the requirements defined by the first step of our research, we will try to identify a suitable hardware description language. If none of existing hardware description languages fully fulfill our needs there will be a need for adaption of one of the existing hardware description languages. Another option is creating a custom hardware description language for purpose of our research.

C. Integration into component models

Once we devise a way to adequately describe hardware devices we must provide an ability to integrate devices into component models.

The first step of this integration is to extend software component description with information about dependencies on hardware devices. Again, one of the main concerns here is how to add this new information

without limiting our ability to reuse components in systems with different hardware configurations. Also, we will have to define how we can create mappings between software components that have dependencies on hardware devices and actual hardware devices specified by our hardware description language.

Besides only including hardware devices when designing component-based systems, we also want to take into account non-functional properties of hardware devices during analysis of such systems. This requires us to also propagate non-functional properties of hardware devices to software components. In order to take advantage of current analysis techniques, we have to present these properties in a way that is usable by these techniques.

D. Evaluation

We will use our research to extend an already existing component model for embedded systems. The whole approach will be evaluated on a number of different case studies with the goal of comparing our approach with methods that are currently used. We will evaluate two properties: analysis accuracy and reuse possibility. As one of our hypothesis is that by adding information about non-functional properties of hardware devices will result in better prediction of system behavior, we will measure if using our approach analysis provides predictions that are more close to real measured values than the values obtained by system analysis that does not take into account hardware devices. Also, we will examine if by applying our approach we will be able to reuse more components between different systems, compared to number of reused components in same systems without using our approach.

VII. Conclusion

In this paper we have illustrated how interaction between software components and hardware devices, such as sensors and actuators, has an important role in component models for embedded systems. In this domain, failure to adequately express dependencies of software components on hardware devices and communication between the two can severely impair our ability to use all benefits that a component-based approach can introduce.

A survey of different component models has shown us that component-based development for embedded systems is still not widely utilized by industry, and most of the component models in this domain are developed and used in the research community. However, lack of any information about how hardware devices are included in component models, and inadequate approaches of some component models that provide this information, leads us to a conclusion that this aspect of embedded systems is still not fully explored when it comes to component-based development.

Therefore, we propose a research plan that will address the lack of proper management of hardware devices in component models for embedded systems. In our research we will provide ability to explicitly state dependencies of software components on hardware devices, define how we can build specification of such devices that will allow them to be incorporated in component models, and provide a way to define mappings between software components and hardware devices. As a result we expect higher reusability of software components and possibility for more accurate analysis.

Acknowledgment

This work was supported by the Unity Through Knowledge Fund via the DICES project, the Swedish Foundation for Strategic Research via the strategic research center PROGRESS, and the Swedish Research Council project CONTESSE (2010-4276).

Reference

[1] AUTOSAR Development Partnership, AUTOSAR – Technical Overview, 2008

[2] Arcticus Systems, http://www.arcticus-systems.com/

[3] Atkinson, C.; Bunse, C.; Gross, H.-G.; Peper, C., "Component-Based Software Development for Embedded Systems – An Overview of Current Research Trends", Lecture Notes in Computer Science, November 2005

[4] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", Adissom-Wesley, 2002

[5] C. Bunse, H.-G. Gross, "Unifying Hardware and Software Components for Embedded Systems Development", In: Architecting Systems with Trustworthy Components, R. Reussner, J. A. Staffort, C. A. Szyperski (Eds), Lecture notes in Computer Science, Vol 3938, Springer, 2006.

[6] Crnković Ivica, Larsson Magnus, "Building Reliable Component-Based Software Systems", Artech House Publishers, 2002

[7] Feiler, P. H., Gluch, D., Hudak, J., "The Architecture Analysis & Design Language (AADL): An Introduction", Software Engineering Institute, Technical Note, CMU/SEI- 2006-TN-011, Feb 2006.

[8] Heinecke H., Damm W., Josko B., Metzner A., Kopetz H., Sangiovanni-Vincentelli A., Di Natale M., Software Components for Reliable Automotive Systems, Design, Automation and Test in Europe, 2008

[9] Juraj Feljan, Luka Lednicki, Josip Maras, Ana Petričić, Ivica Crnković, "Classification and Survey of Component Models", DICES technical report, University of Zagreb, 2009, available at http://www.fer.hr/dices/

[10] Kaj Hänninen, Jukka Mäki-Turja, Mikael Nolin, Mats Lindberg, John Lundbäck, Kurt-Lennart Lundbäck, The Rubus Component Model for Resource Constrained Real-Time Systems, 3rd IEEE International Symposium on Industrial Embedded Systems, 2008

[11] Ke Xu, Sierszecki Krzysztof, Angelov Christo, COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems, RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007

[12] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The SAVE approach to component-based development of vehicular systems. Journal of Systems and Software, 80(5):655– 667, May 2007.

[13] OMG. SysML Version 1.1, 2008.

[14] R. van Ommering, F. van der Linden, and J. Kramer. The Koala component model for consumer electronics software. In IEEE Computer, pages 78–85. IEEE, March 2000.

[15] T. Bureš, J. Carlson, I. Crnković, S. Sentilles, and A. Vulgarakis, "ProCom – the Progress Component Model Reference Manual, version 1.0," Mälardalen University, Technical Report MDH-MRTC-230/2008-1-SE, June 2008.