MICOBS: Multi-Platform Multi-Model Component Based Software Development Framework *

Pablo Parra
University of Alcala
Alcala de Henares, Spain
parra@aut.uah.es

O. R. Polo
University of Alcala
Alcala de Henares, Spain
opolo@aut.uah.es

Martin Knoblauch
University of Alcala
Alcala de Henares, Spain
martin@aut.uah.es

Ignacio Garcia
University of Alcala
Alcala de Henares, Spain
ngarcia@aut.uah.es

Sebastian Sanchez
University of Alcala
Alcala de Henares, Spain
ssp@aut.uah.es

ABSTRACT

This paper presents a framework designed to work with a multi-platform approach over two levels of definition of an embedded system built from software components. In the upper level, the framework focuses on the composition of components and the analysis of extra-functional properties following the principles of compositionality and composability. In the lower level, the framework provides a packaging model aimed to automate the tasks of configuration and construction of an executable out of the software bundles that constitute the components' implementation. For both levels, the framework introduces the concept of platform as a new dimension on the specification that will affect every transformation carried out between the models integrated in the framework. This dimension copes with the fact that a component has a large amount of extra-functional properties that are inherently dependent on the platform on which the component is finally deployed. The framework also provides a multi-model support. Firstly, it allows building systems from already implemented components conforming to existing component models (CCM, SOFA, FRACTAL, etc.). Secondly, it enables the definition of new abstract component models that, though lacking a native implementation, can be used to develop specific applications using model-driven engineering processes. Finally, the framework allows the integration, in a single system, of heterogeneous components conforming to different component models. An example of this integration, with components coming from MyCCM-HI and EDROOM models, is presented as a use case.

*This work was supported by MICINN under the project AYA2009-13478-C02-02.

CBSE'11, June 20–24, 2011, Boulder, Colorado, USA.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Design

Keywords

Embedded Systems, Component-Based Software Engineering, Model-Driven Engineering

1. INTRODUCTION

Component-Based Software Engineering (CBSE) [16] is a powerful way of software reuse inspired in the success of the hardware design approach from IP cores and commercial-off-the-shelf (COTS) components. A software component can have different implementations depending on the platform on which it is deployed and, most important, its Extra-Functional Properties (EFPs) may vary depending on that platform. An example of this can be the Worst Case Execution Time (WCET) of a given function or the memory size occupied by the corresponding compiled object. This feature makes it necessary to have development environments that introduce the platform as a new dimension. This dimension would allow the developer to specify different implementations and to assign different values of the EFP to a component linked to a given platform. With this approach, each *multi-platform* component would be managed as a classical component during the system's construction but hiding a platform indexed element organization that would contain multiple realizations and their associated information for the analysis. During the EFP analysis of the different deployment alternatives prior to the final system's deployment, the corresponding analysis information could be properly selected and managed depending on the different candidate platforms.

After analysing the current software component solutions for embedded systems, a new issue arises due to the difficulty to integrate in a single system several heterogeneous components that have been created with different development tools, and might even be supported by different component

model implementations. This kind of incompatibilities reduces the chances for reuse, forcing component developers to maintain several versions of one single component, each one cast to a particular type of implementation, thus hindering the efficiency of the component development approach. Regarding this, the existence of multi-model development environments seems to be more than adequate. These environments must provide a component meta-model that hides the underlying component model implementation. By virtue of this abstraction, these environments should also provide the ability to create new systems through the integration of heterogeneous components supported by different component model implementations.

Finally, it is necessary to consider that the benefits of component based modelling can also be extended to specific application domains adopting the Model-Driven Engineering (MDE) [1] approach. In this kind of solutions the semantics inherent to the problem to be solved are generally expressed through models of a much higher level of abstraction than those typically offered by regular component models. Building native component implementations for this kind of models is a time-consuming and error-prone solution. Therefore, it is more adequate to implement them by means of transformations from the abstract models to component models that are directly supported by a component model implementation.

The MICOBS framework presented in this paper proposes a solution to the three aforementioned issues. To this end, it includes the concept of platform as a specification variable and, based on this concept, it offers support for the creation of component libraries where each component features implementations associated to different platforms and incorporates evaluation information of the EFPs for each of them. MICOBS enables the use of these components on a composition model named Multi-platform Component Architecture and Deployment (MCAD). MCAD allows the architectural definition of every new system, facilitating the specification of different deployment alternatives and their subsequent analysis. This analysis is based on the EFPs of the associated elements depending on the platform selected in each case. In addition, MICOBS provides multi-model support in the sense that it allows working with different component models even within the same system. This feature is supported by the artefact named *component domain*. A component domain is defined by a component model, a set of model-checking operations, and a set of model-to-model (M2M) and model-to-text (M2T) transformations. MICOBS provides a common meta-model in order to define each domain's component model. The framework infrastructure linked to the meta-model automatically provides a default set of model-checking operations for each domain component model. These operations can be redefined if needed to reflect particular semantics of the given domain. Transformations, in turn, provide, among other services, the capability to import already implemented components using an external toolset (such as Save-IDE [15], ASSERT/TASTE [8], EDROOM CASE Tool [19] or the tools provided by FRACTAL implementations such as Julia or Cecilia [2]). It also allows the definition of inverse transformations to perform the full exporting to an external toolset of a system's MCAD model already deployed on a specific platform.

MICOBS defines two domain types: *implementation oriented domains* (IODs), which have a reference component model implementation, such as SaveCCM [7], CORBA Component Model (CCM) [12], FRACTAL, ASSERT/TASTE, Koala [18]; SOFA/2 [3] or EDROOM; and *application oriented domains* (AODs), which enable MICOBS to support an MDE process using application-specific component models with a higher level of abstraction. AODs require transformations to IODs to obtain the system's implementation. MICOBS also allows defining a special subtype of IODs, called *complex* IODs, which enable the creation of models that integrate heterogeneous components coming from other IODs into a single system. A complex IOD design pattern is provided by the framework in order to reduce the effort of building implementations that support this integration. When a new development project for building an embedded system is initiated, its MCAD model will have to determine on which domain it will be built. The corresponding domain model-checking operations will be used to supervise and verify the construction of MCAD models and, in the case of complex domains, it will also verify that the integration of heterogeneous components follow the rules established by the domain.

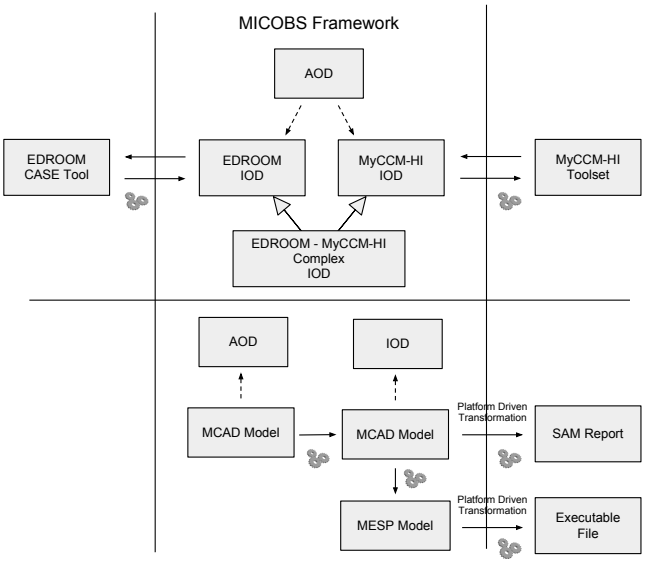

Figure 1: Scheme of the MICOBS framework

With respect to the analysis process, MICOBS manages the EFPs organizing them in *system analysis models* (SAMs). The purpose of each SAM is to obtain an EFP analysis report out of a system's MCAD model. To this end, the elements of the MCAD model will be decorated with the values of their EFPs in consonance with a set of Analysis Oriented Models (AOMs) that will specify the rules on how the values have to be assigned. Apart from the set of AOMs, the SAM must also implement the transformation that will, based on these models, generate the corresponding analysis report.

Finally, with the purpose of automating the development process of a whole embedded system, the framework undertakes the tasks of configuring and constructing the final executable out of the software bundles that constitute the components' implementation. In order to achieve this, a packaging level has been defined below the composition level of the MCAD model and its domains, supported by

a model called Multi-platform Embedded Software Packaging (MESP). This model also features the concept of multi-platform development, allowing the grouping of all the implementations of a component for the different platforms in a single software bundle. Figure 1 summarizes the concepts exposed in this introduction regarding the role of the different elements that compose the structure of MICOBS. The upper half figure shows how the domains are organized and an example of how, through the IODs, it is possible to integrate two external toolsets into the framework. The lower half focuses on the MCAD models definition and how, through automatic transformations, it is possible to obtain the system's executable and the SAM analysis reports.

In the following sections we describe in detail all the information related to this work. Section 2 introduces the platform concept and its role on the transformations aimed to obtain the final products. Section 3 describes how to use MICOBS in the context of a Component Engineering Process. This information includes the definition of the actors who take part and determines their workflow. Sections 4 and 5 detail, respectively, the two levels of the framework's internal structure: the composition level and the packaging level. Section 6 describes an evaluation example of the framework where the construction of a complex IOD is explained; Section 7 presents related works; and the last section shows some conclusions and future works.

2. THE PLATFORM CONCEPT

The MICOBS Framework defines a platform as the tuple P = (osapi, os, architecture, microprocessor, board), where:

- *osapi*: identifies the interface provided to the applications by the operating system. E.g. POSIX.
- *os*: identifies the operating system that supports the embedded application. E.g. RTEMS.
- *architecture*: identifies the instruction set architecture (ISA) implemented by the microprocessor that will execute the embedded application. E.g. SPARC.
- *microprocessor*: identifies the microprocessor on top of which the application is to be executed. This microprocessor implements the ISA identified by the architecture element of the platform. E.g. ERC32.
- *board*: identifies the specific board supporting the embedded system. This board includes the microprocessor identified by the previous element. E.g. Development Board or Engineering Model.

The platform concept is present on every model that is part of the MICOBS framework, but it does not add any complexity to the modelling tasks. After the modelling, the working platform is selected, and through different transformations the framework will allow the obtaining of a set of final products, such as the final executable file of the application, different models for analysis or models compatible with other external toolsets.

3. MICOBS COMPONENT ENGINEERING PROCESS

The use of the MICOBS supports an engineering process for defining, designing and implementing component-based embedded systems. This process involves the following actors:

- *Framework Architect* (FA): creates new component domains and integrates them in the framework. It also defines the SAMs, with their related AOEMs, and implements the transformations in order to automatize the analysis report generation.
- *Component Provider* (CP): is in charge of defining new components and adding them to a component library corresponding to the domain. It also completes the AOEMs description of each component for the different deployment platforms.
- *Component Tester* (CT): designs and implements the component test procedures and manages the testing platforms.
- *Application Architect* (AA): uses the set of components defined in the component library to design the MCAD model of an embedded application. It analyses the SAMs reports of the deployment alternatives in order to chose a suitable platform that meets the design metrics [17]. If the application requires the definition of more components or the refinement of the existing ones, the AA should perform the request to the CP.
- *Application Developer* (AD): manages the application deployment using the transformations provided by the domain.
- *Application Tester* (AT): designs, implements and executes the validation tests for the developed application and manages the testing platforms.

A scheme of the actor's roles inside this process is shown in Figure 2.

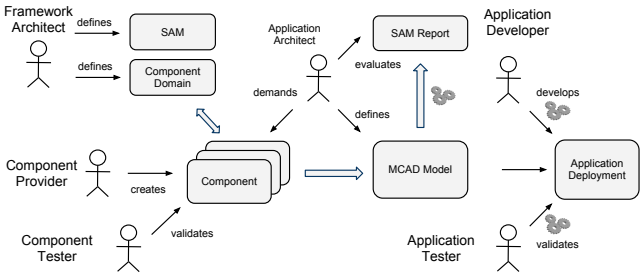

Figure 2: MICOBS Component Engineering Process

4. MICOBS COMPOSITION LEVEL

Within the features provided by MICOBS, this level is focused in the definition of new components and its associated domains, and the building, based on these components, of the system architecture and deployment model. This level also adds the possibility of defining the SAMs and its related AOMs that enable the assignment of the EFPs to the component model elements in order to obtain the SAMs reports. Finally, it also manages the integration into the framework of external toolsets; the possibility of carrying out MDE processes by means of AODs; and the definition of heterogeneous systems using complex IODs. Next subsections describe these features in detail.

4.1 Component Domain Insight

As we have already mentioned in the introduction, a component domain is defined by a component model and a set of model-checking operations and model transformations. MICOBS provides a meta-model to define the component model belonging to the domain, expressing the semantics of the different basic elements and the relationships between them. Within this meta-model, a *component* is defined as a compositable and deployable unit that can be instantiated and that provides and requires services to and from other component instances by using communication points called *ports*. These ports can adopt one out of two roles: *client* or *server*. A client port demands the services provided by a server port. Ports are linked to an *interface*, which defines the set of services demanded or provided. Component instances can be bound by setting *connections*. A connection is always a bind between a client port and a server port. A *service library* is defined as a stateless and linkable entity that can provide services to the component instances and can also demand services from other services libraries. The meta-model defines the *service access interface* (SAI) as a set of semantically-related services and uses this abstraction to define the dependency relationships between a component and a service library and between the service libraries themselves. The meta-model is built under the following assumptions:

- Hierarchical component models are allowed. A containment relationship can be established between a component (called *parent*) and one or more internal instances of other components.
- Components can have external and internal ports. The external ports are the ones offered to the rest of the components. The internal ports are the communication points used by the component to communicate with its internal instances.
- A relay connection can be established between a port of an internal instance and an external port of its parent. This connection is feasible only if both ports have the same type, interface and role.
- Component models can support singleton component instances.

The meta-model diagram is shown in Figure 3. Its basic elements are the typifiers `MComponentType`, `MPortType` and `MInterfaceType`, along with the elements that they typify, called respectively: `MComponent`, `MPort` and `MInterface`. The elements `MComponent`, `MPort`, `MInterface` and `MConnection` are representations of component, port, interface and connection respectively. `MConnector` is equivalent to connector. Finally `MSAI` and `MServiceLibrary` are the same as SAI and service library.

The following elements are an example of the definition of the CCM using the meta-model provided by MICOBS:

- An `MComponentType` called `CCMComponent`, corresponding to a monolithically implemented component. This component is not allowed to instantiate sub-components.
- Another `MComponentType` called `CCMComponentAssembly`, corresponding to an assembly of component instances. Components of this kind can instantiate inside of them another components of both this type and the `CCMComponent` one.
- Two `MPortType`, called `CCMAsynchronousPort` and `CCMSyncrhonousPort`, corresponding to the ports implementing asynchronous and synchronous communications respectively.
- Two `MConnector`, called `CCMAsynchronousConnector` and `CCMSynchronousConnector`, to perform the connections between asynchronous ports and synchronous ports respectively.
- Two `MInterfaceType`, called `CCMSyncInterface` and `CCMAsyncInterface`. The first one corresponds to the methods of the synchronous interfaces, and the other one to the asynchronous events. OMG's IDL will be used to define the different interfaces.

After the component model has been defined, the domain is completed by the implementation of a series of operations organized in two different sets: model-checking operations and transformations. The model-checking operations provide restrictions to the Component Editor for the components definition and also to the MCAD Model Editor that defines the embedded system architecture. By default, they are automatically generated based on the relationships established between the component model elements. The framework architect can customize these functions to define more complex constraints to the component model. These functions will be used by the Component Description Editor to validate the component descriptions and during the MCAD model definition.

Meanwhile the transformation operations enable the automation of the MICOBS process workflow. They can be model-to-model, model-to-text or a combination of both. The framework architect can choose to define which operation it will provide depending on the level of integration achieved by the domain's implementation. This topic will be further discussed in Section 4.4.

4.2 MICOBS Component Common Representation

The MICOBS framework provides a common language, called Component Common Representation (CCR), to describe the components defined according to a component model within a given domain. The CCR enables the integration of the components with the rest of the elements and tools of the framework composition level. Based on each domain component model, the Component Description Editor produces the CCR for the components associated to that domain. The language provides enough semantics to be able to describe the main characteristics of the components:

- The architectural description of the component, i.e. the internal or external ports through which the instances of the given component will provide or demand services to and from other instances.
- The hierarchical composition of a component.
- The platforms supported by the implementation of the component.
- For each supported platform, the SAIs required by the implementation that must be provided by service libraries.

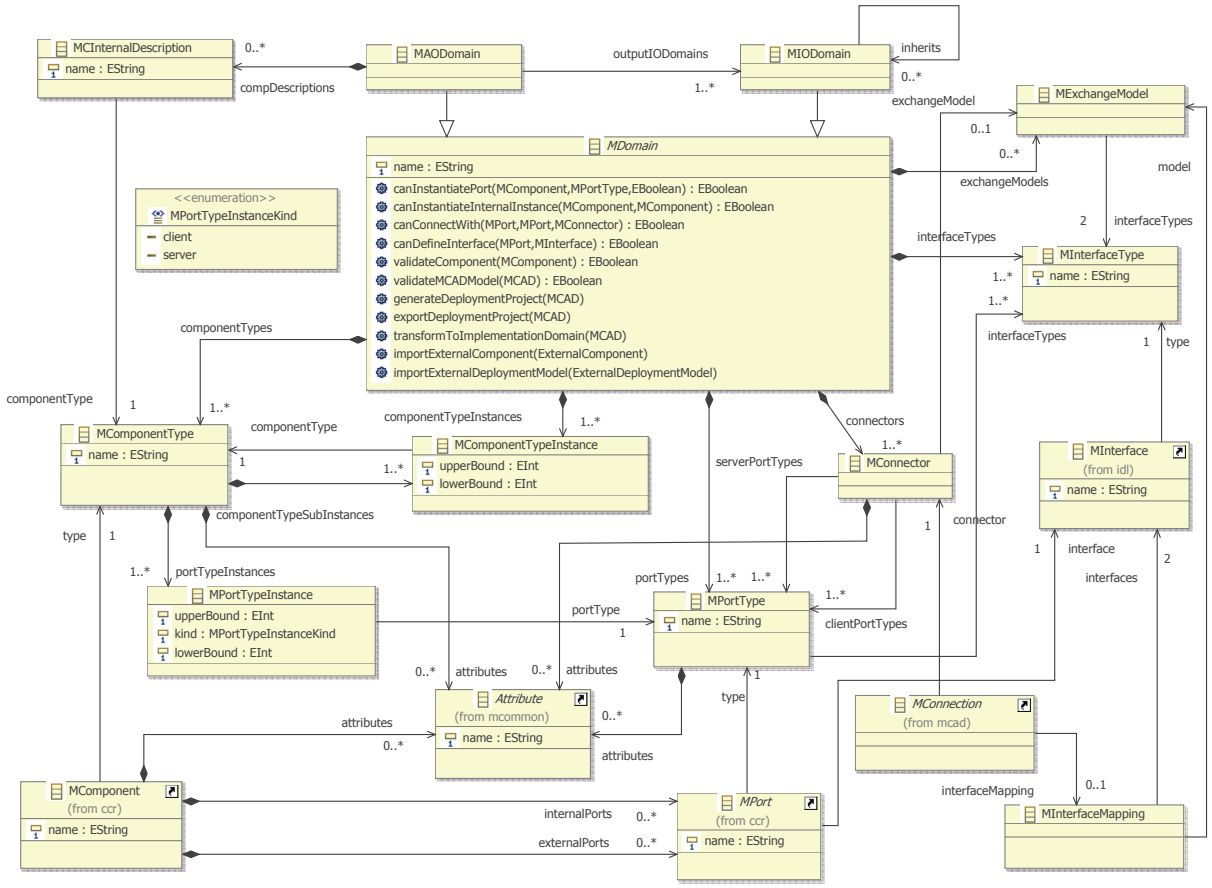

Figure 3: Simplified component meta-model diagram

- The value of the attributes defined by the component type.
- The specification of the attributes that will parametrize the component instances.

The architectural description of the component is divided into two membranes: one external and one internal. The external membrane groups all the external ports of the component, while the internal does the same with the internal ports. The CCR allows the description of the different internal component instances, as well as their connections with themselves and with the component's internal membrane. The instances and the connections can be specified depending on the supported platforms of the parent component, which means that the internal structure of the component can be made dependant on the platform on which it is deployed. This internal structure is divided in a platform independent area, where the component can define component internal instances that will be deployed independent of the deployment platform, and one area for each of the supported platforms.

4.3 The MCAD Model

The MCAD model is used to design the architecture of the application and customize its deployment on the different platforms, and is used within the framework as an input for obtaining several output products. Depending on the degree of functionality implemented by the corresponding domain, these outputs can include: the final executable file or a mean to obtain it if the domain is implementation oriented; different analysis reports corresponding to the SAMs supported by the model; and, when the domain is application oriented, another MCAD model belonging to a different domain. An MCAD model is defined as a set of component instances and their connections, defining the system architecture and allowing the definition of multiple nested deployment alternatives. A deployment alternative represents a possible component deployment, and defines a list of component instances, service libraries, connections and possibly another nested subalternatives.

An MCAD model is defined as a set of component instances and their connections, defining the system architecture and allowing to define multiple nested deployment alternatives. A deployment alternative represents a possible component deployment, and defines a list of component instances, service libraries, connections and eventually another nested subalternatives.

The set of nested deployment alternatives can be seen as a tree, with each node representing one alternative, and its branches each of its subalternatives. From the set of leaf nodes the framework can infer which are the possible deployment alternatives of the system.

Connections can be established between the component instances of an alternative and the instances of its subalter-

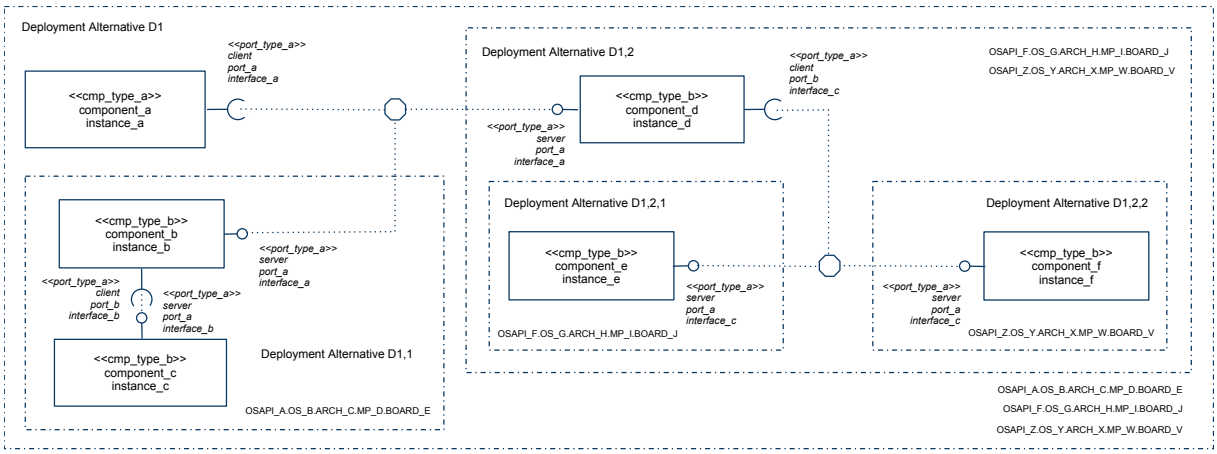

Figure 4: Example of an MCAD model

natives using connection switches. When the MCAD model is fully defined, a set of final deployments can be created from the different combinations of the leaf node alternatives that share the same supported platforms. This artefact has two main purposes: to define different solutions for implementing the same functionality and to perform a multi-alternative multi-platform deployment.

In the example of Figure 4, we have three leaf-noded deployment alternatives, i.e. D1.1, D1.2.1 and D1.2.2. Each one represents the system's deployment on a different platform. As we can see, depending on the chosen platform, we will have a different component architecture. The application architect will be able to obtain, from this MCAD model, three different analysis reports, one for each valid alternative combination, that will give him a valuable insight on the characteristics that will be found on the final deployed system.

4.4 Domain types

Domains can be classified according to the outputs generated from the MCAD model. If the domain transformations are intended to generate the final executable file over an specific component model implementation, then the domain is said to be *implementation oriented domain* (IOD). On the other hand, if the domain enables a model-driven engineering process using an application specific high level abstraction component model, then it is called an *application oriented domain* (AOD). AODs require always transformations to IOD models in order to get the final system implementation.

Implementation oriented domains

Within this kind of domains, the elements have a direct relationship to software elements. An example of an IOD would be the one given in Section 4.1, which defines CCM as its component model. Depending on the level of tool integration achieved with the implementation oriented domain's definition, there are two possible paths or options to perform the deployment of the application. If the transformation provided by the domain to the *Application Developer* is able to perform the full process without needing of any other external tool from outside the framework, then the domain is a highly integrated domain. On the contrary, if the Application Developer needs of any additional functionality provided by an external tool, then the domain is called a *loosely inegrated domain*. Highly integrated domains implement an operation to generate, from the MCAD model, all the necessary artefacts to create a deployment project. These artefacts include the code of the connectors and the rest of the platform's support code. The code of the components, along with the generated connector's code, will be in the form of MESP software resources (see Section 5). A MESP model is created with the resulting software resources that include all the necessary information to configure, generate and optionally deploy the final executable file.

Loosely integrated domains implement an operation to generate an exchange model. This model is used as an input of an external toolset that will be in charge of performing the final steps in the process of obtaining the final executable file. This model could be, for example, a deployment plan model conforming to the OMG's Deployment & Configuration specification.

Application oriented domains

The components belonging to an AOD are intended to be abstract representations of the application elements that do not have a biunivocal relationship with a component model implementation. Furthermore, they need a series of transformations to translate them to one or more software components. These mechanisms are a key part in the support provided by the framework to development component based methodologies driven by model transformations. An AOD can be extended with one or more component internal description models. These models can be used as an input for model-to-model and model-to-text transformations to obtain other components or an actual implementation of a part or the whole of the component. MICOBS does not dictate what language or methodology is to be used to define these models but the Ecore model from the Eclipse Modelling Framework (EMF) [6] is strongly recommended because of its easy integration with MICOBS. The definitions should be attached to a given `MComponentType` defined within an AOD.

An AOD can define element types that do not correspond to an actual component implementation but do need a transformation to obtain that implementation. For example, an

AOD can define an `MComponentType` whose behaviour and structure and, thus, whose implementation is defined using an internal description model. In our example, we have an embedded system corresponding to the on-board computer of a research satellite. This satellite is in charge, among other things, of executing certain experiments on a set of instruments and to send the results back to the ground station. An internal model will be defined to specify the set of experiments, their execution patterns and the format of the information that has to be sent to ground. Using this internal model, part of the code of the software component, conforming to the actual component model that is used to develop the whole on-board software can be generated using M2M and M2T transformations defined on the AOD.

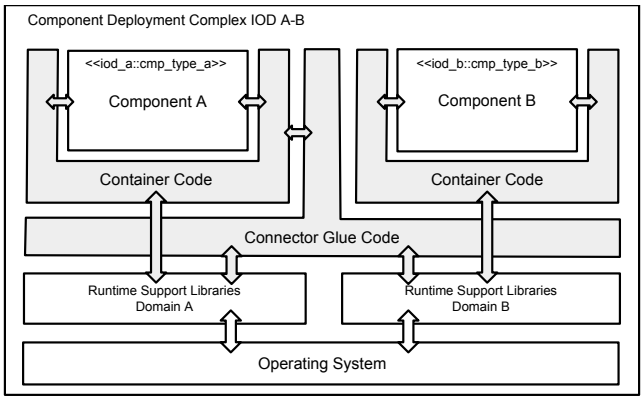

Figure 5: Complex IOD implementation schema

4.5 Coping with heterogeneity

As we have previously mentioned, one of the objectives of the framework is to support the integration of components conforming to different models and implementations. In order to do so, we have defined a concept called complex IODs. The framework architect can create complex IODs as a combination of two or more IODs. The mechanism that MICOBS provides for this feature is domain inheritance, allowing a complex IOD to inherit from one or more parent IODs. By means of this relationship, the new complex IOD will include all the types and connectors defined by its parent domains. The framework architect can include new types and connectors to define the different communication means to interconnect the heterogeneous components. He will also have to define new model-checking operations and new transformations to generate the implementation of the connectors. MICOBS does not dictate how the complex IODs should be implemented. However, a design pattern based on the definition of new connectors is encouraged to maximize the reuse of the previous component implementations. Within this approach, new connectors are defined to perform all the communication tasks between the heterogeneous components. These connectors are attached to exchange models that define how the mappings between the different interfaces have to be done. The connector's code will be automatically generated using model-to-text transformations based on those interface mappings, which are linked to the corresponding connection. A template of this kind of transformation for the complex IODs connectors is provided by MICOBS as a reusable design pattern. Apart from the connector's code, the domain might also need to provide code to adapt the component's implementations to the connectors. A diagram with the elements involved in the process is shown in Figure 5, with the grey parts indicating the automatically generated code.

On a general basis, this approach will be feasible if the components implement their own tasks as most of the component model implementations do. If on the contrary the runtime environment manages the execution of the components, it is most likely that the component model implementations of the parent domains will not be compatible or cannot be linked or executed together, and that a new runtime environment should be implemented for the subdomain that will support the implementations of all the components. The example in Section 6 shows how a complex domain implementation has been achieved for EDROOM and MyCCM-HI component IODs.

4.6 Analysis Models

Based on the composability and compositionality principles, MICOBS can generate different reports to analyse different properties of a built system. In order to provide this feature, the framework manages an updateable set of *system analysis models* (SAMs). A SAM is composed of a set of *analysis oriented models* (AOMs) that are defined according to the system analysis semantics and are able to capture both EFPs as well as internal descriptions of the different elements of the MCAD model. MICOBS do not impose any limitation on which language should be used to specify the AOMs. There are two kinds of AOMs: *analysis oriented element models* (AOEM), which can be associated to the different elements of the MCAD model of the deployed system (domains, components, ports, connections, service libraries, etc.); and *analysis oriented platform models* (AOPM), which define properties about the platforms over which the system is deployed. The AOEMs are, in turn, divided in:

- *Platform Independent AOEMs*: the information stored in that model will be independent of the platform on which that particular element can be deployed. These models can store, for example, behavioural information of the elements.
- *Platform Dependent AOEMs*: the information will depend on the platform on which the element can be deployed. E.g. footprint, power consumption, etc.

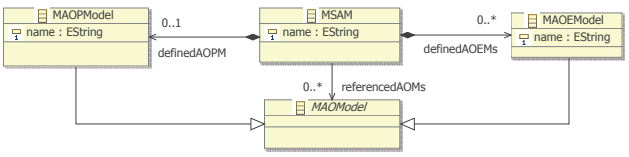

Figure 6: SAM meta-model definition

One SAM can define any amount of AOEMs and, optionally, one AOPM. Also, it can reference external AOMs defined on other SAMs. The meta-model of a SAM is shown in Figure 6. The architect has to implement one operation to generate the analysis report which will be the input of an existing tool, such as UPPAAL or SPIN model-checkers, or of a custom-made tool developed by the architect.

SAMs are domain independent, meaning that they are not restricted to a particular domain component model. An example of an EFP that could be analysed is the *footprint*, which is clearly platform dependent. We have used EMF's Ecore to define the different AOMs and QVT Operational to generate the analysis report. Our approach has been affected by a special characteristic present in the vast majority of the libraries used with embedded software; the possibility of performing conditional linking. This means that the linker will only add the library functions that are actually used by the software to the final executable file, not the whole library object file. This fact complicates the assignment of the footprint property to the service libraries, since the actual amount of memory used will depend on the services actually required by the instantiated components and connectors, as well as on the ones required by other services libraries. We have to be able to model this to obtain an accurate analysis on the memory footprint. We have defined two SAMs. The first SAM, called `SAMDependencies` contains three platform independent AOEMs. The first AOEM is used to model the service access points (SAPs) of a MSAI. A SAP represents a single service that can be isolated and identified from within an MSAI, e.g. a function belonging to a full application interface. An MSAI is attached, using this first AOEM to a list of its corresponding SAPs. The second AOEM is used to model the dependencies between the SAPs of the provided and required MSAIs of a given `MServiceLibrary`. Each SAP of the provided MSAIs is linked to the list of SAPs of the corresponding required MSAIs on which they depend. The third AOEM is used to identify within an `MComponent` or an `MConnector` which SAPs of all the required MSAIs are actually used.

The second SAM, called `SAMFootprint`, defines the AOMs needed to attach the information related to the memory footprint to the different model elements. It defines three models. The first model is a platform dependent AOEM used to define the set of internal functions present in an `MServiceLibrary` and their memory footprint depending on the platform and to establish the relationship between these functions and the provided SAPs. Like this, we are able to obtain the functions that are going to be linked when a given SAP is demanded and, furthermore, their footprint. The second model is another platform dependent AOEM used to specify the footprint of an `MComponent` or an `MConnector`. This footprint is directly assigned depending on the platform. The third and last model is an AOPM, which specifies the footprint of the operating system and its system libraries associated to the platform. Figure 7 shows the meta-model of the first AOEM. Once the AOMs are defined, the model for the final analysis report and the transformations needed to obtain it have to be defined. Using the AOEMs linked to the MCAD model elements, we obtain all the dependencies between the MSAIs and its corresponding SAPs and, thus, we can generate the lists of internal elements that are actually used of every `MServiceLibrary` that is linked on the corresponding deployment. This information is independent of the platform where the library runs. From this information and the information stored in the AOMs regarding the footprint of the operating system and the `MComponents` and `MConnectors`, we can obtain the final calculation on the amount of memory that will be needed to deploy the application defined in the MCAD model.

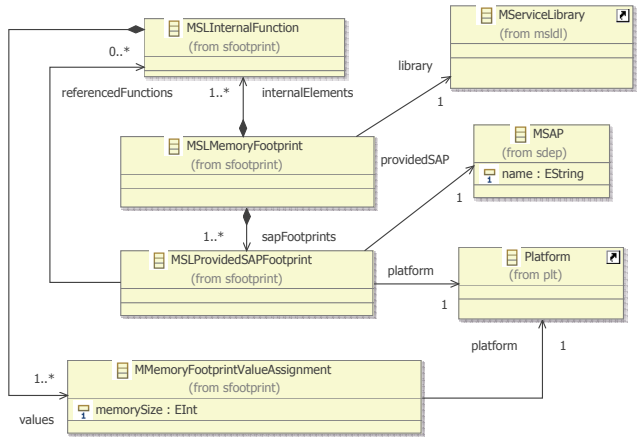

Figure 7: Meta-model of the AOEM used to assign footprint values to an MServiceLibrary

5. THE MICOBS PACKAGING LEVEL

The composition level provides a suitable abstraction to the analysis and system design stages. Nevertheless, the implementation and the final deployment stages require support for both software packaging and system configuration. To provide this support MICOBS provides a model called MESP and a set of transformations, extending it towards the multi-platform approach present in the whole framework. The elements of the MESP model are the `MSwResource`, the `MSwInterface` and the `MPlatformBundle`. The `MSwResource` is defined as a software package that can be supported for one or more platforms and that contains all the elements and all the necessary information to allow it to be integrated with other packages to form the final application's executable image. An `MSwResource` provides and eventually requires `MSwInterfaces`. Depending on the programming language on which the resources are implemented, the `MSwInterfaces` can refer from a set of functions or definition to more complex structures, like classes. The MESP model contains two specific elements, called `MOSSwResource` and `MOSSwInterface`. The first element represents the resource associated to the operating system and the second, the set of functions, data types and definitions that conform the application programming interface (API) provided by the operating system used in the embedded system. The last of the elements is the `MPlatformBundle`. It stores all the necessary information to generate the final executable file from the different software resources. It is linked to a specific platform. A MESP model contains the references to all resources that conform the final application. Following the same approach we used to define the MCAD model, the MESP model also allows the definition of multi-alternative and multi-platform deployments. MICOBS provides a toolset to obtain, from this model, a MESP deployment project.

6. EVALUATION

We have successfully implemented this framework as a set of Eclipse Plug-ins, using the technologies provided by the EMF. We have defined a series of Ecore models to define each of the models of the framework and used them to create editors using the Xtext language modeling tools [6]. For each domain, the framework architect defines a new

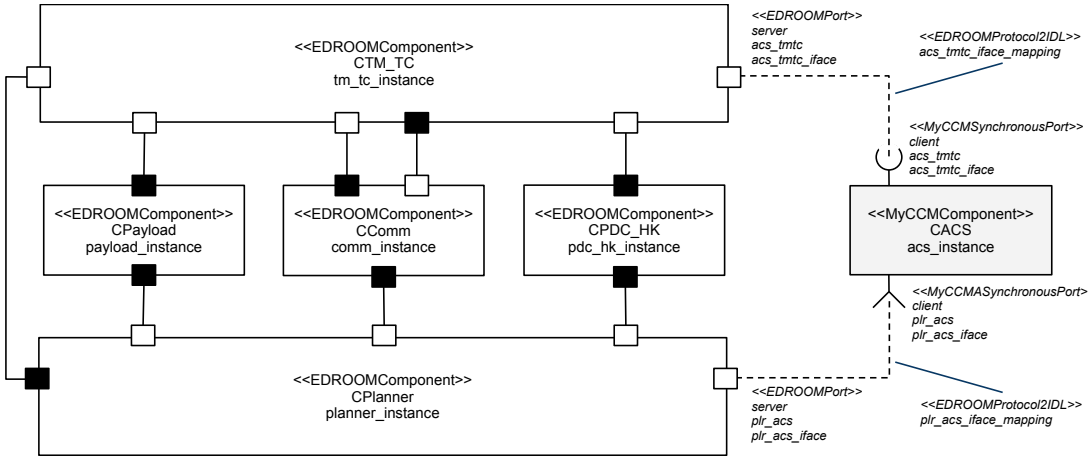

Figure 8: Satellite on-board application built on the MyCCM-HI-EDROOM complex IOD

Eclipse plug-in that will contain not only the description of the domain but also the Java implementation of the operations described in Section 4.2. The methodology used to implement the model translation has been to define transformation using the QVT Operational Language. For model-to-text transformations, we have chosen to use Xpand [6]. In our study, we have defined two simple IODs, one corresponding to the EDROOM component model and one to the MyCCM-HI component model. Finally we have evaluated the coping with heterogeneity feature of MICOBS defining a complex IOD, resulting from EDROOM and MyCCM-HI IOD integration.

An example of a satellite on-board application, built on this complex IOD, is shown in Figure 8. This system is built as a cooperation of five EDROOM components plus a MyCCM component, called `CACS`, which is in charge of the Attitude Control System (ACS). This component receives the satellite's target attitude from the component `CPlanner`, the value of which can be updated during the satellite's mission. `CACS` implements a periodic task that performs all the necessary calculations, based on the information received from the sensors, and programs the satellite's actuators to modify its position according to the target attitude. This process can also generate telemetry information to be sent to ground, which will be handled by the `CTM_TC` component.

Two `MConnector` elements of the complex IOD, called `MyCCMAsyncEDROOM` and `MyCCMSyncEDROOM`, are able to connect an `EDROOMPort` with a `MyCCMAsynchronousPort` or a `MyCCMAsynchronousPort` respectively. We have defined an exchange model to be able to specify the mappings between the interfaces defined using the two interface description languages, i.e. OMG's IDL files and EDROOM Protocol descriptions and to perform their adaptation.

From the MCAD model, the implementation of different connectors is automatically generated using XPand templates and the corresponding interface mapping linked to the specific connection. This implementation uses functions belonging to the support libraries of both component model implementations to perform the message translation and to manage the message queues used by the connectors. Apart from the connectors, it also generates the implementation code for the static deployment of all the components.

7. RELATED WORK

Several component models, such as CCM, OpenCOM, JavaBeans, or FRACTAL do not address the possibility of obtaining any kind of analysis model from the property description of the different elements [4]. MICOBS can provide them the possibility of performing an analysis process. SaveCCM provides the possibility of defining the behaviour of their modelled components to perform statical analysis at design time [9]. In this case, MICOBS can provide a mechanism to extend them with new analysis models and multi-platform specification management of both the functional and extra functional properties.

The OMG's D&C specifies a platform independent model to perform the deployment and configuration of component-based applications. Being mainly focused in distributed environments and, more specifically, in CCM, it does not provide any default mechanism to annotate the different model elements with EFP, nor specify the type of connection mechanism or connector used to connect two components, although extensions have been suggested to provide the latter functionality [13]. Ada-CCM [10] is a framework based on OMG's D&C and Lightweight CCM that provides a component model and its implementation, adding new semantics to the original models in order to be able to perform real-time schedulability analysis. This complex functionality, models and tools are limited to that particular component model, not being able to extend it to other component models. CoSMIC [14] is a tool that applies the OMG's MDA to the deployment and configuration of component based applications following the D&C specification, reducing the complexity of performing these tasks. It also provides mechanisms to perform analysis at design-time of certain real-time aspects related mainly to QoS. This analysis however is done per-application, which means that it does not use any compositional mechanism to obtain the analysis model.

ACME [5] ADL does provide semantics for element annotation, including connectors. These annotations are limited to property assignments and constraints, and are not expressed depending on the platform or the implementation. Furthermore, both D&C and ACME lack of the possibility of performing an alternative-based multi-platform deployment.

Apart from the previous features, MICOBS also allows the definition of complex implementation domains and AODs,

integrating MDE techniques in the development of component based software systems.

ROBOCOP [11] is an example of a component model that supports property analyses depending on the platform. It defines the components as an extensible set of models representing different features or characteristics, e.g. functional models, demanding resource models, executable models, etc.; allowing the possibility of defining relationships between them. MICOBS is also based on models and model transformations to perform the analysis but its approach is to provide a framework capable of being used on a broad set of component technologies.

8. CONCLUSIONS

Component-based modelling for embedded systems provides a mechanism for software reuse. In this paper we have presented MICOBS, a multi-platform multi-model framework intended to help in the development of component-based embedded systems. It defines the deployment platform as a new dimension on the system's specification. This dimension drives the M2M and M2T transformations that manage the workflow of the whole framework development process and lets the developer to specify different implementations and to assign different values of the EFP to a component linked to a given platform.

The framework allows the composition of heterogeneous components conforming to different component models, as well, as to define abstract component models that can be used to develop specific applications using MDE processes. Finally, the integration of components from the MyCCM-HI and EDROOM models, has been presented as a use case. Furthermore, MICOBS will be used, within the Space Research Group of the University of Alcala, to develop the on-board software of the MICROSAT spacecraft of the Spanish National Institute of Aerospatial Technology (INTA).

9. REFERENCES

[1] C. Atkinson and T. K. Model-driven development: A metamodeling foundation. *IEEE Software*, 20.

[2] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36:1257–1284, September 2006.

[3] T. Bures, P. Hnetynka, and F. Plasil. Sofa 2.0: Balancing advanced features in a hierarchical component model. In *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 40–48, Washington, DC, USA, 2006. IEEE Computer Society.

[4] I. Crnković, S. Sentilles, A. Vulgarakis, and M. R. Chaudron. A classification framework for software component models. *IEEE Transactions on Software Engineering*, 99(PrePrints).

[5] D. Garlan, R. Monroe, and D. Wile. Acme: an architecture description interchange language. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '97, pages 7–. IBM Press, 1997.

[6] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 1 edition, 2009.

[7] H. Hansson, M. Akerholm, I. Crnković, and M. Torngren. Saveccm - a component model for safety-critical real-time systems. In *Proceedings of the 30th EUROMICRO Conference*, pages 627–635, Washington, DC, USA, 2004. IEEE Computer Society.

[8] J. Hugues, M. Perrotin, and T. Tsiodras. Using mde for the rapid prototyping of space critical systems. In *Proceedings of the 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, Washington, DC, USA, 2008. IEEE Computer Society.

[9] M. kerholm, 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. *J. Syst. Softw.*, 80:655–667, May 2007.

[10] P. López Martínez, J. M. Drake, P. Pacheco, and J. L. Medina. Ada-ccm: Component-based technology for distributed real-time systems. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, CBSE '08, pages 334–350, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] H. Maaskant. *A Robust Component Model for Consumer Electronic Products*, volume 3 of *Philips Research*. Springer Netherlands, 2005.

[12] Object Management Group. *CORBA Component Model (CCM), v.3.1.* OMG document formal/08-01-04.

[13] S. Robert, A. Radermacher, V. Seignole, S. Gérard, V. Watine, and F. Terrier. The corba connector model. In *Proceedings of the 5th international workshop on Software engineering and middleware*, SEM '05, pages 76–82, New York, NY, USA, 2005. ACM.

[14] D. C. Schmidt, A. Gokhale, R. Natarajan, E. Neema, T. Bapty, J. Parsons, J. Gray, A. Nechypurenko, and N. Wang. CoSMIC: An mda generative tool for distributed real-time and embdedded component middleware and applications. In *In Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*. ACM, 2002.

[15] S. Sentilles, A. Pettersson, D. Nystrom, T. Nolte, P. Pettersson, and I. Crnković. Save-ide - a tool for design, analysis and implementation of component-based embedded systems. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 607–610, Washington, DC, USA, 2009. IEEE Computer Society.

[16] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[17] F. Vahid and T. Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.

[18] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The koala component model for consumer electronics software. *Computer*, 33, March 2000.

[19] A. Viana, O. R. Polo, O. López, M. Knoblauch, S. Sánchez, and D. Meziat. EDROOM: A free tool for the uml2 component based design and automatic code generation of tiny embedded real time systems. In *Proceedings of the 3rd European Congress on Embedded Real-Time Software ERTS*, 2006.