# A Component-based UML Profile to Model Embedded Real-Time Systems Designed by the MDA Approach

Shourong Lu and Wolfgang A. Halang
Fernuniversität
Faculty of Electrical and Computer Engineering
58084 Hagen, Germany
shourong.lu@fernuni-hagen.de

Lichen Zhang
Guangdong University of Technology
Faculty of Computer Science
510090 Guangzhou, China
lchzhang@gdut.edu.cn

## Abstract

*A component-based UML profile is built to develop embedded real-time systems. To specify the specific characteristics of embedded systems, the Model Driven Architecture (MDA) approach and component-based modeling are employed. Component-based UML models are designed as Platform Independent Models (PIM) to be translated to Platform Specific Models (PSM) for target-platform implementation, which deal with functional and non-functional properties. Taking specific platform features into regard, specific component models result from transformations mapping a platform-independent component model to either the Process and Experiment Automation Real-Time Language (PEARL) or to Function Blocks according to IEC 61131-3 or IEC 61499. Both PIM and PSM are collected in a UML profile which can be used as an application framework in developing embedded real-time systems.*

## 1 Introduction

Embedded real-time systems contain a computer as a part of a larger system and interact directly with external devices. Taking into account all constraints, system design has to fulfill demanding requirements with respect to limited resources, real-time requirements, reliability, cost and re-usability. In other words, in developing such systems one has to consider non-functional properties, because the correct operation of a system is not only dependent on the correct functional working of its components, but also dependent on its non-functional properties.

Nowadays, component-based development appears to be an attractive approach in the domain of embedded systems. It is expected that it could bring a number of advantages to the embedded systems world, such as rapid development times, the ability to re-use existing components, and the ability to compose sophisticated software [10]. But the contemporary component models are not suitable to develop embedded real-time systems, since they do not address issues such as timeliness, quality-of-service or non-functional and predictability properties that are highly important for embedded real-time systems. Furthermore, they are inherently heavyweight and complex, which leads to large overheads on the run-time platforms, and they do not support dynamic reconfiguration of components. Therefore, it is indispensable to develop a new model meeting the requirements of embedded real-time systems.

The Unified Modeling Language (UML 2.0) [9] offers an unprecedented opportunity to describe component-based embedded systems. It provides constructs to deal with varying levels of modeling abstraction to visualise and specify both the static and dynamic aspects of systems. And with the mechanisms stereotypes, tagged values and constraints the semantics of model elements can be customised and extended. The Object Management Group (OMG) has proposed the Model Driven Architecture (MDA) approach, which aims to allow developers to create systems entirely with models. This approach envisages systems to be comprised of many small, manageable models rather than of one gigantic monolithic model, and allows systems to be designed independently of the technologies they will eventually be deployed on. It is based on two essential concepts, viz., the Platform Independent Model (PIM) and the Platform Specific Model (PSM), which separates the specification of system functionality from the specification of the implementation of that functionality on a technology-specific platform, and provides a set of guidelines to structure specifications expressed as models [6].

Therefore, it is useful and significant to combine the MDA approach and UML models with the component technique to develop software for embedded systems. The paper is organised as follows. Section 2 describes model driven approaches, basic component-based modeling technique in

Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)
1533-2306/05 $20.00 © 2005 IEEE

order to provide a basic understanding. Section 3 and Section 4 present how to design a PIM and PSMs component model employing the MDA approach and UML notations based on component-based modeling. Finally, Section 5 concludes the component-based UML models.

## 2 MDA and Component-based Modeling

**Platform Independent Model (PIM)** is a highly abstracted model independent of any implementation technology. It captures the essential features of a system, and specifies what the system does. It may include generic functions, scenarios, and classes to describe how the system realises its requirements. A PIM can be mapped into one or more PSMs. **Platform Specific Model (PSM)** specifies how the system is implemented. It determines how the PIM executes in a target deployment environment. Such a PSM describes in detail how the PIM is implemented on a specific platform, or in a certain technology. **Platform Model** is the final product, corresponding to code written in a specific programming language for a specific application.

Component modeling deals with three kind of models, namely, **Structural Models** consist of components, classes, and their relationships. It represents the static configuration of a system through the dependencies and connections between components. **Behavioural Models** is used to describe the dynamic aspects of the components, component interaction and resource constraints. It can be divided into component-interaction parts that show the messages (behavioural name and/or message argument) sent between components (or subcomponents or classes), and state transition parts that present the state transitions of each component or the interactions between the components. **Functional Models** specify how operations derive output values from input values without regard to the order of computations.

## 3 PIM Component-based UML Model

Following the approaches of MDA and component-based modeling, an embedded real-time system's architecture is specified by setting up PIM and PSM.

As shown in Fig. 1, a platform-independent component model for embedded real-time systems is defined, which is described by defining the set of stereotypes meeting the specific requirements in the application domain, such as Active Component, Passive Component, and Event Component. Each of them has own attributes. Component is responsible for the functional aspects, while Connector is responsible for the control and communication aspects of a system. A set of Ports is assigned to Component, and Roles are assigned to Connector. These ports and roles are the interfaces of components and connectors, and obey exactly one

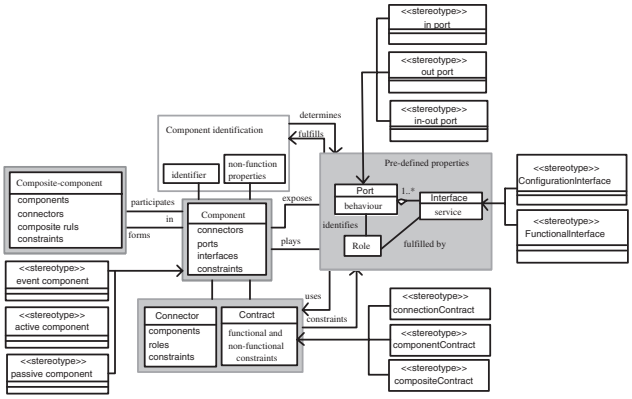


**Figure 1. Platform independent model**

protocol which specifies the order of incoming and outgoing messages. It can be formulated by a 7-tuple of the form [8]:

---

$CBUM = (n, BSC, CSC, ST, CL, SIG, I)$

- $n$ : is a unique name of the software component;
- $BSC$ : (BasicComponent) is a set of software components;
- $CSC$ : (CompositeComponent) is a set of software components;
- $STC$ : (StereotypeComponent) is a set of stereotype components;
- $CL$ : (ComponentClass) = $MC \cup CC$ where:
  $MC$ : member class (set of classes)
  $CC$ : common class (set of classes)
  $CC = Met \cup Atb$ where:
  $Met$ : method class (set of classes)
  $Atb$ : attribute class (set of classes)
- $SIG$ is a set of 6-tuple signatures describing operations (functions):
  $SIG = (nf, In, Local, Out, Pre, Post)$ where:
  $nf$ : name of operation (function)
  $In$ : list of input parameters
  $Local$ : list of local variables
  $Out$ : list of output parameters
  $Pre$ : expression
  $Post$ : expression
- $I$ is a set of 3-tuple signatures describing transactions:
  $I = (IS, ID, Bh)$ where:
  $Is$ : interaction source
  $Id$ : interaction destination
  $Bh$ : is a set of 3-tuple behaviour parts describing the states and actions between $Is$ and $Id$:
  $Bh = (nb, St, Ac)$ where:
  $nb$ : name of behaviour
  $St$ : set of states
  $Ac$ : set of actions

---

## 4 Platform-specific Component Models

As shown in Fig. 2, here we consider two transformations from a platform-independent component model to platform-specific component models. One is a mapping from PIM to the Process and Experiment Automation Real-Time Language (PEARL) [2], the other one is to function blocks as defined in IEC 61131-3 [5]. For each transformation, some rules need to be defined to generate PSMs from a PIM.



Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)
1533-2306/05 $20.00 © 2005 IEEE

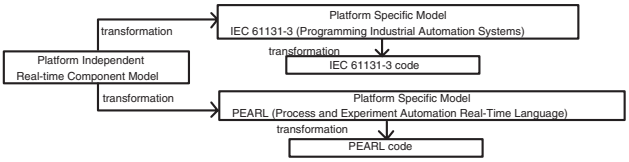


**Figure 2. PIM to PSM transfomations**

### 4.1 Transformation from PIM to PEARL

Before we transform a PIM component model to PEARL, we first give a short introduction on PEARL to provide a basic understanding. PEARL is one of the very few genuine high-level real-time languages. With industrial real-time applications in mind, its development began in the late 1960s by a group of industrial companies and research institutes. Incorporating decades of experience, its later version PEARL90 was standardised in 1998 [2], and its extension for distributed systems in 1989 [1]. PEARL has been extended towards safe object-oriented and distributed applications (PEARL*, Hi-PEARL*, Safe PEARL*, Verifiable PEARL*) [4]. Owing to its clear concepts, combining features from classical programming languages and special ones for real-time systems, PEARL is particularly suitable for industrial process control and embedded applications.

PEARL for distributed systems includes elements to describe hardware and software configurations. An architecture description consists of *station division*, *configuration division*, *net division*, and *system division*, which describe different associated layers of a system design. A more detailed description can be found in [3].

Now we return to the **model transformation**, whose essential point is the mapping to the PEARL architecture, its real-time features, and its run-time constraints. It is performed in two separate steps, the first one being a transition between PIM components and PEARL nodes. Here as an example we only map to PEARL *collections* (architecture allocation). The second step is to assign attributes to *collections* (attribute assignment). The transformation rules are defined as: (1) The platform-independent component model is implemented as a PEARL collection having to meet real-time constraints. (2) For each class in the PIM component model, a PEARL-oriented stereotype is defined according to PEARL architecture features. (3) The timing attributes of a *task* in PEARL can be defined as *TaggedValues* and assigned to corresponding stereotypes in the platform-specific model. (4) The constraints of real-time tasks can be described by *Constraints* expressed in the Object Constraints Language, and applied to port and connector stereotypes of the platform-specific model.

By parameterisation and attribute assignment, the transformed component model can be expressed with the corresponding stereotypes depicted in the following and summarised in Fig. 3.

```
Context PStation inv:
 inv: self.baseClass = Node
 self.ownedElement.IsInstantiable = true
 self.ownedElement.contents -> forAll (m|
    m.OclIsKindOf(componentInstance) and
    m.OclIsKindOf(port) and
    m.OclIsKindOf(connectors) and
    m.OclIsKindOf(rules))

Context PCollection inv::
 inv: self.baseClass = component
 self.ownedElement.IsInstantiable = true
 self.ownedElement.contents -> forAll (m|
    m.OclIsKindOf(Module) and
    m.OclIsKindOf(Port) and
    m.OclIsKindOf(Task) and
    m.OclIsKindOf(Line))

 Context PConnection inv:
 self.baseClass = Connector and
 self.ownedConnection -> forAll(ac|
 ac.TaggedValue -> exists(tv:TaggedValue|
   tv.name = "role" and
   tv.value.oclIsKindOf(Set(port))and
   tv.value -> size >=2 ))

 Context PPort inv:
  self.baseClass = Port and
  self.ownedPort -> forAll(oclIsTypeof(PEARProt))
  self.ownedRole -> forAll(oclIsTypeof(ProtocolRoles))
  ProtocolRoles:
  inOperation :: outOperation :: in/outOperation
  Protocol: blocking-send :: send-reply :: no-wait-send
```

| PEARL Element | UML Element | Stereotype | Icon |
|---|---|---|---|
| Station | Node | <<PStation>> | |
| Component | Component | <<PComponent>> | Workstore, Device, Proctype |
| Connection | Connector | <<PConnection>> | |
| Collection | Component | <<PCollection>> | |
| Port | PortClass | <<PPort>> | |
| Module | Class | <<PModule>> | |
| Task | Class | <<PTask>> | |
| Line | Class | <<PLine>> | |

**Figure 3. Stereotypes for PEARL**

### 4.2 Transformation from PIM to Function Blocks

Function blocks are defined within the standard's IEC 61131-3 software model [5] and IEC 61499 [7].

As shown in Fig. 4, an instance of a function block is specified by input, output, in-out and internal variables, and by an internal behaviour. The input variables can only be written from the outside of the FB. From the inside they can be read, only. Output variables can be read and written



Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)
1533-2306/05 $20.00 © 2005 IEEE

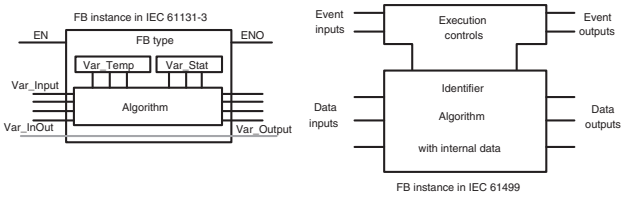


**Figure 4. FB instances**

from the FB's inside, and only be read from the outside. In-out variables are special shared variables. If their data types match, output variables can be connected to input variables by connectors. Assignment of data values to interface variables is the only means to communicate with other FBs. Function blocks have internal state information that persist the execution of FB instances. A characteristic feature of FBs is the separation of external interfaces and internal implementation. In IEC 61499, the function block concept is more related to execution control and scheduling, because IEC 61499 addresses distributed control systems, in which it is very important that software modules residing on different devices follow a well predictable execution sequence. Periodic or event-driven real-time tasks can be associated with function blocks.

The rules for **model transformation** to generate from a platform-independent component model a PSM meeting platform features of the function block model are: (1) The platform-independent component model is mapped to a function block-oriented model by defining corresponding stereotypes, such as FBActive, FBPassive and FBEvent. (2) The ports in the component model are direct counterparts of the interface variables of FBs. The input and output variables are mapped to in-ports or out-ports, respectively, and the in-out variables to in-out ports to which a Contract (with matching protocols) is added. The stereotypes FBInPort and FBOutPort are defined as examples as shown below and formulated in OCL. FBOutPort requires one FBInterface, and FBInPort provides one FBInterface. The FBInterface is needed to model input and output variables of FBs; it may contain operations. The FBInterfaces can be mapped to dataInterface and controlInterface of a component according to its operations.

```
context PBInterface inv:
    self.baseClass = Interface,
     self.ownedElement.contains->forAll( m|
          m.OclIsKindOf(InputVariables) and
          m.OclIsKindOf(OutputVariables))

context FBOutPort
 inv: self.baseClass = Port
   self.provided->size = 0 and
   self.required->size = 1 and
    self.required->forAll( r |
           (r = Interface))

context FBInPort
inv: self.baseClass = Port
       self.required->size = 0 and
       self.provided->size = 1 and
       self.provided->forAll(p |
            (p = Interface))
```

In addition to the defined in-ports, out-ports, in-out-ports (IEC 61131-3) stereotypes, there is still need to define Data Port, Control Port and Event Port (matching the IEC 61499 paradigm) stereotypes as follows according to the requirements of function blocks.

## 5 Conclusion

The architecture specification of an embedded system works with with two models, viz., PIM and PSMs. The PIM component model captures the essential features of the system, and specifies what it is to do, while a PSM component model describes how the PIM component model is implemented on a specific platform. Hence, obvious benefit are low cost by fostering re-usability of old designs (PIM) in new applications (PSM), and valuable when validating the correctness of models, and facilitates to model the features of embedded real-time systems.

### Acknowledgments

This work was supported by a STIBET Matching Funds grant of both DAAD and ifak e.V. (S. Lu), and in part by grant no. 60474072 of the National Natural Science Foundation of China (L. Zhang).

## References

[1] DIN 66253 Teil 3: Mehrrechner-PEARL. Berlin-Köln: Beuth Verlag 1989.

[2] DIN 66253-2: *Programmiersprache PEARL90.* Berlin-Köln: Beuth Verlag 1998. D.S. Frankel: *Model Driven Archtecture*. Addison-Wesley 2003.

[3] R. Gumzej: *Embedded System Architecture Co-Design and its Validation.* Doctoral thesis, University of Maribor, 1999.

[4] W.A. Halang, C.E. Pereira and A.H. Frigeri: Safe Object Oriented Programming of Distributed Real Time Systems in PEARL. Proc. *4th IEEE Intl. Symp. on Object-Oriented Real-Time Distributed Computing*, pp. 87 – 94. Los Alamitos: IEEE Computer Society Press 2001.

[5] IEC 61131-3: *Programmable Controllers, Part 3: Programming Languages*. Geneva: International Electrotechnical Commission 1992.

[6] A. Kleppe, J. Warmer and W. Bast: *MDA explained: The model driven architecture: practice and promise*. Addison-Wesley 2003.

[7] R. Lewis: *Modelling control systems using IEC 61499, Applying function blocks to distributed systems*. IEE Control Engineering Series, No. 59, 2001.

[8] K.-K. Lau and M. Ornaghi: A Formal Approach to Software Component Specification. Proc. *Specification and Verification of Component-Based Systems*, 2001.

[9] Object Management Group: *Unified Modeling Language: Superstructure*. OMG document ptc/2003-08-02, 2003.

[10] C. Szyperski, D. Gruntz and S. Murer: *Component Software — Beyond Object-Oriented Programming*. 2nd ed. Addison-Wesley 2002.



Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)
1533-2306/05 $20.00 © 2005 IEEE