Functional and Structural Properties in the Model-Driven Engineering Approach

Daniela Cancila
DIMI Università degli Studi di Udine, Italy
CEA LISE Saclay Paris France
cancila@math.unipd.it

Roberto Passerone
DISI Università degli Studi di Trento, Italy
roberto.passerone@unitn.it

Abstract

In this paper we discuss the separation between attributes on functionality and on structure following an approach based on Model Driven Engineering (MDE). We adopt a methodological approach based on correctness-by-construction for modeling high-integrity real-time embedded systems. We illustrate how this separation is implemented by a prototype, recently realized by our research team. Software reuse is incremented by using the prototype. This has been confirmed by the evaluation of two teams from major European space industry. We conclude our work by discussing some open problems.

1 Introduction

The benefits of software reuse in terms of increased productivity have attracted the attention of the scientific and industrial community in recent decades, with a special focus on component-based design techniques [7, 4, 27, 17, 20, 33, 32, 13]. In this context, new methodologies such as Model Driven Engineering (MDE) [27] and Platform-Based Design (PBD) [26], have been proposed, in which the different aspects of a design are considered separately for efficiency, and only later brought together according to precise rules that provide certain guarantees of correctness. This separation of concerns allows designers to develop specifications for function, timing, architecture and structure which can later be reused or adapted to more quickly achieve the desired result.

Along these lines, researchers are devoting much effort towards the identification, specification and implementation of *attributes* on functionality and structure in the design development process. Broadly speaking, attributes on functionality have to do with the functional specification of methods, interfaces and classes. Attributes on structure, instead, regard all non-functional features, from concurrency management to security guarantee. For this reason, they are also known in the literature as non-functional attributes [16, 5, 24]. This separation is expected to provide an additional increase in software reuse: the more a functional specification of a system is free from structural aspects, the more it can be reused and/or integrated in another functional specification. Two examples of application of these concepts are the ASSERT European project [1] and, more recently, the IMOFIS French project.

The main result of this paper is a conceptual separation between attributes on the functionality and on structure in the MDE approach in light of correctness-by-construction (CbyC) [14], intended as the limitation of the designer's freedom to only specify "correct architectures" of system components. As a result, designer cannot introduce errors on the architecture. (See Section 2.1).

This separation is based on solid mathematical foundations, which we sketch in Section 4.3 (see [10] for more details). Such a separation has been implemented in two views, one included in the other, in a prototype for high-integrity real time systems, by following the approach introduced in [33]. The prototype has been developed by our research team at the University of Padova (Italy) and has been successfully tested by two teams from major European space industry.

1.1 Related Work

In the literature, attributes on the functionality are usually specified using formalisms based on UML2. But for attributes on structure, we are faced with a changing and non-homogeneous landscape. For example, Franch and Botella subdivide non-functional attributes into three macro categories [16]. The first category is about time and space, reusability, maintainability and reliability, and is defined according to the guidelines introduced by standardization bodies [19]. The second category concerns the behavior and is identified with ports, connectors and components. Finally, the last category is about "non-functional requirements" viewed as software unit.

Quality of Service (QoS) attributes are taken under consideration in the component-based approach introduced by Bruni et al. [7]. The authors introduce and discuss Architecture Design Rewriting (ADR), which is based on graph theory and term rewriting. The notion of architecture is intended as the (possibly dynamic) structure of a network of components. QoS attributes are viewed as parameters of the connections between com-

1-4244-1506-3/08/$25.00 ©2008 IEEE

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

ponents and are modeled as edges in the network. More in particular, they are considered as "soft" constraints, i.e., a constraint which can be satisfied with a certain quality. Term rewriting is used to describe "legal" reconfigurations of the network.

When the component-based design approach coincides with the model-driven approach [27, 28], then the identification, specification and implementation of attributes on structure becomes more complex, since a single modeling space is substituted with more modeling spaces (i.e., the platform independent model, or PIM, and the platform specific model, or PSM). In most cases, the difference between PIM and PSM reflects the separation between functional and structural models [22, 21]. The functional model abstracts from the requirements of the implementation. The structural model, instead, addresses the requirements placed on the run-time behavior to be exhibited on the target platform and it must also undergo the verification and validation activities that precede deployment. In this context, attributes on functionality and on structure have to be specified and managed.

Röttger and Zschaler adopt a model-driven architecture approach [24]. Attributes on structure capture QoS and security aspects. The authors focus on QoS only with particular attention to response time, delay and memory usage. Designers give a first functional representation of a system by using primarily UML2. Then, s/he refines the given representation by non-functional attributes, which carry time and space measurements on connectors and ports and add new model elements which decompose the functionality of a component in several subcomponents. The whole process is iteratively refined at different levels of abstraction. The proposed methodology is influenced by a hierarchical top-down approach to software design. However, the top-down approach has the (questionable) property of violating the principle of encapsulation [30, 17]. It is not clear how the approach discussed in [24] resolves the violation of encapsulation.

In our work, we follow the approach proposed by Vardanega [33] and, more in general, the one developed within the ASSERT EU Project [1]. Unlike the previous work described above, in our approach we build a tool by starting from a mathematical formalization of a computational model which is particularly suitable to study high-integrity real-time systems [8, 34, 10]. Moreover, such a formalization and the tool integrate the MDE approach with CbyC. The impact of the theoretical and methodological study on real applications with the development of supporting software is also considered by the teams which study BIP, Metropolis, AADL and MARTE [4, 3, 25, 23]. A first comparison in this direction is given in [10, 6].

In the following sections we will first define the area of the problem and present in Section 3 a real example of application which includes a description of our adopted technology. Section 4 discusses our approach in more detail, followed by the industrial assessment.

2 Scope and Overview

We are interested in modeling high-integrity real-time embedded systems, in which extreme attention is placed on the ability to deliver provable evidence of predictability and correctness. To delimit the scope of our work, we assume that the functional specification of each component is correct, i.e., the component does what it is supposed to. Instead, we focus on how components are aggregated to form a system and investigate, in particular, the issue of "structural correctness". Our methodology follows the MDE approach, where components are composed in a platform independent model (PIM), and then realized in a platform specific model (PSM). Thus, the problems we are mostly concerned with are, for instance, related to the kind of run-time entities that may exist with respect to the adopted platform, the attributes that should be specified on the components at the PIM level for a correct implementation, and how to guarantee that model transformations from PIM to PSM preserve the local properties of the components.

To address these problems, we adopt the *Ravenscar Computational Model* (RCM) [34, 33]. RCM provides a model space that is compatible with the restrictions of the Ada Ravenscar profile [8]. It is aimed at addressing two crucial questions for high-integrity real-time systems: how to manage concurrency through trustworthy architectural choices, and how to guarantee static analyzability of a system. Because of these features, RCM may be regarded as an attractive model space for the design of this kind of embedded systems.

The functional attributes are specified by designers with respect to the functional requirements. These include, for instance, the signature of the methods and their visibility. In oder to increase reuse, the functional specification of a system should be independent from both the platform and the technology. Likewise, the designer specifies attributes related to the structure of the system. These attributes are of two kinds: attributes on performance metrics, e.g., deadline of a process, and attributes on the intended behavior at run-time, for example a periodic activation of a flow of control or a mutually exclusive access to a resource. Unlike functional attributes, structural attributes are not completely independent from run-time entities. In the model-driven approach, components in the designer's space (PIM) are automatically transformed in run-time components (PSM). As a result, some information must be "promoted" from the PSM to the PIM, so that a designer can "correctly" specify the components by selecting attributes on the system structure. This is similar to a platfrom-based design approach, where an abstract version of the lower level platform is exported to the higher levels of abstraction to facilitate mapping and performance evaluation. As a consequence, the run-time entities at the PSM level should also be able to separate functional and structural attributes, without altering their semantics. This particular case of correctness-by-

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

construction is outlined in [33].

2.1 Assumptions of CbyC

In [33], CbyC is defined with respect to architectural correctness of a high-integrity real-time embedded system. In the sequel, with the terms correctness and correct, we mean architectural correctness and architecturally correct. CbyC must be defined to guarantee that:

- a designer specifies only correct components;
- the correctness of a component is preserved by model transformations;
- the composition of individually correct components preserves the correctness of each component;
- the correctness of the system is directly derived by the operation of composition.

3 Example and Methodology

Figure 1 shows an example studied in the ASSERT project [1]. The system, a simple abstraction of a satellite, is composed of four components: POS (short for position store); GNC (guidance, navigation and control); TMTC (telemetry and telecommand); and PRO (propulsion). POS is a data storage, used in R/W mode by GNC and TMTC. TMTC acts according to one of the two commands that it may receive from the outside environment: it either writes to POS the value specified in the received telecommand, or sends PRO a boost correction request (also specified in the received telecommand). GNC performs a periodic feedback-control loop on POS: it first reads the current value; it then computes the required adjustment; and it finally updates the initial value accordingly. The operation of GNC is locally correct if and only if the value written to POS is consistent with the value read from POS at the beginning of the operation. PRO periodically executes either a default operation or else the command required by TMTC.

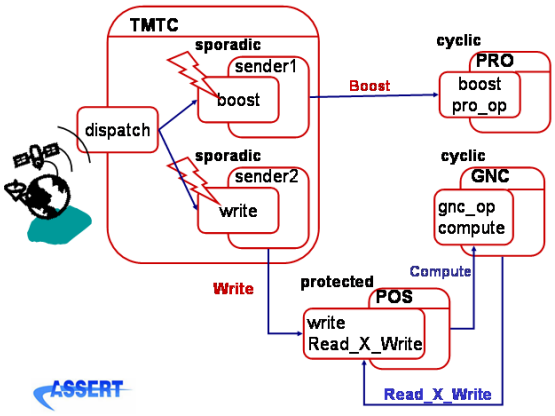

Figure 1. Simple satellite system

There are three aspects that need to be addressed. The first is how to achieve a correct management of concurrency. We want to abstract from ad-hoc solutions and be based on architectural choices. Second, we wish to separate the specification of functional attributes of a component, for example the visibility of the methods, from the specification on the structure, for example the deadline of flow of control. We need therefore to understand which attributes are of type functional and which of type structural and how a designer can manipulate them. Finally, we need to guarantee that the automatic model transformation from PIM to PSM realizes the intended meaning of the specification.

In order to manage the concurrency, RCM adopts a priority-based preemptive scheduling, based on the Priority Ceiling Protocol [18, 29]. In an RCM system, all resources are statically allocated. The flows of control do not interact directly but through (protected) resources, which provide a mutually-exclusive access to data. Priority is statically assigned to each flow of control, so that the flow with the highest priority is scheduled for execution. The priority of a protected resource that is shared between two or more flows of control is greater than or equal to the highest priority of all calling flows of control. When a flow of control invokes a protected resource, it acquires the priority of the protected resource, also called the *active priority*. Consequently, it cannot be preempted by other flows which use the same protected resource. In our example, POS has priority greater than or equal to the priority of both GNC and TMTC components. This policy avoids deadlocks and provides an optimal bound for the worst-case duration of priority inversion [29, 34].

Functional and structural attributes are attached to each single method of an RCM component. RCM de-structures an interface (as classically intended, for example a UML interface) into elementary interfaces, each of them formed by one single method. An elementary interface of an RCM component can be either *provided* (PI) if it defines the provided contractual obligations associated with the service, or *required* (RI) if it defines the services which are required from other components to fulfill the contractual obligations. By de-structuring the interface we can annotate each method with attributes that specify its intended run-time implementation. For example, the same method can be decorated as *cyclic*, if it is periodically activated by a hardware interrupt from the execution environment, or *sporadic*, if it is sporadically activated by a call coming from either a software component or a hardware interrupt. The cyclic or sporadic attribute forces the realization of the associated method by a cyclic or a sporadic RCM run-time components. Tables 1 and 2 show the list of possible low-level functional and structural attributes that can be specified for each PI and RI. In addition, when $\mathcal{W} = 1$, additional parameters are required to specify timing requirement such as deadlines, activation period, minimum separation between successive sporadic activations and priority.

User-level attributes are defined as particular combinations of the low-level attributes. For example, the cyclic attribute corresponds to having $\mathcal{S}$ = avoidance synchro-

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

Functional	Meaning
Profile	method signature
State	static variables operated on by the method
Visibility $\mathcal{V}$	public, private, restricted
Structural	**Meaning**
C_l	worst-case execution time, *excluding* the cost of all invoked RI
C_g	worst-case execution time *including* the cost of all invoked RI (automatically calculated)
Ceiling	priority ceiling (automatically calculated)
$\mathcal{S}$	synchronization guarantee: none ($\mathcal{S} = \emptyset$); exclusion synchronization; avoidance synchronization.
$\mathcal{W}$	control flow: execution in caller thread ($\mathcal{W} = 0$); execution in callee thread ($\mathcal{W} = 1$)
τ	$\tau = RI_1, \ldots, RI_k$, guarantees transactional access to state(s) across the execution of all of the invoked RIs; $\tau = 0$ no guarantee

Table 1. Meaning of attributes on a PI

Functional	Meaning
Invocations	number of times that the RI may be invoked in the execution of the PI
Structural	**Meaning**
C_g	maximum duration that the execution of the invoked PI may take *including* the cost of any operations invoked by the RIs exposed by that PI (automatically determined)

Table 2. Meaning of attributes on an RI

nization, $\mathcal{W} = 1$, $\mathcal{V} =$ private, $\tau = 0$. The period is specified as an additional value of the $\mathcal{W}$ parameter. The combination of attributes that are compliant to RCM [8, 34] are discussed in [12].

The correctness of model transformations from PIM to PSM has been discussed in [12]. The underlying idea is the following. An RCM run-time component is an indivisible entity resulting from an aggregation of five elements: provided and required interfaces (PI, RI); a functional part that represents the functional and sequential specification associated to a single PI and the RI needed to fulfill it; at most one synchronization protocol which guarantees mutual-exclusive and state-dependent access to internal data or no guarantee at all; at most one flow of control. The same set of attributes (Tables 1 and 2) is applied at both the PIM level (RCM components) and the PSM level (RCM run-time components). These attributes influence how the five elements are aggregated together and which

Attribute	Meaning
cyclic	PI periodically invoked by hardware interrupt. $\mathcal{S} =$ avoidance synchronization, $\mathcal{W} = 1$, $\mathcal{V} = private$, $\tau = 0$.
sporadic	PI sporadically invoked by hardware or software interrupt. $\mathcal{S} =$ avoidance synchronization, $\mathcal{W} = 1$, $\mathcal{V} = any$, $\tau = 0$.
modifier	PI sporadically or periodically invoked by hardware or software interrupt. It changes the behavior of, and it is executed by the flow of control associated with either a cyclic or a sporadic operation. $\mathcal{S} =$ avoidance synchronization, $\mathcal{W} = 1$, $\mathcal{V} = any$, $\tau = 0$.
protected	PI which provides a mutually-exclusive access to data, executing in the caller thread. $\mathcal{S} =$ mutually exclusive access, $\mathcal{W} = 0$, $\mathcal{V} = any$, $\tau = 0$.
transactional	PI which provides a mutually-exclusive access to data, executing in the caller thread, without interruption. $\mathcal{S} =$ mutually exclusive access, $\mathcal{W} = 0$, $\mathcal{V} = any$, $\tau = 1$.
passive	PI which provides no guarantee to access to data, executing in the caller thread. $\mathcal{S} =$ empty, $\mathcal{W} = 0$, $\mathcal{V} = any$, $\tau = 0$.

Table 3. User-level attributes

RCM run-time entity is generated. A designer has then a powerful way to influence RCM run-time components, without the need to know the details of their correct aggregation.

3.1 Tool support

As part of the ASSERT EU project [1], our research team at the University of Padova has realized a software tool prototype, based on [33], to design high-integrity real-time embedded systems. The objectives of the prototype were to: (1) provide a software design development tool based on the model-driven engineering approach, (2) provide a design environment which guarantees correctness-by-construction; (3) statically validate a system together with a round-trip engineering process which automatically propagates the results of the analysis throughout the entire model, from views in PSM to views in PIM; (4) automate certain operations to facilitate designers who do not necessarily know the RCM methodology, without penalizing experienced users.

Goal (2) has been widely discussed in the ASSERT technical documentations and more recently in [12, 10]. Goals (3) and (4) concern technological choices. The editor is a plug-in for Eclipse [15], an open-source framework realized in Java. There are two main advantages in using Eclipse: a wide academic and industrial community, and a large number of plug-ins which can be freely used.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

Among them: EMF, to formalize the metamodels; GEF to define graphical diagrams; ATL to specify model transformation and MOFscript to automatically generate code.

EMF uses a formalism based on Ecore. A model in Ecore has approximately the same expressiveness of a UML2 diagram: a designer can create classes, their relationship, signature, and so on. Constraints are expressed using OCL [35]. In addition, EMF exploits XMI Ecore files and gives the ability to generate a plug-in for Eclipse to create models using an API. Finally, EMF provides an editor with a tree-representation. In order to have a graphical representation we have used GMF, since it combines features of both EMF and GEF, and the generated code can easily be enriched with results from the analysis.

4 Solution

To simplify our presentation we discuss a simplified version of the tool. In particular, we focus on the relationship between the Functional view and the Interface view together with their realization in the Implementation view. The prototype provides additional views. A guide is available in the RCM tutorial [11].

4.1 Functional view

In the Functional view, the designer specifies the functional services provided by system components and expresses their sequential behavior in terms of classes and interfaces by using a language similar to UML2. An example of Functional view is shown in Figure 2 (the dispatcher is omitted for lack of space). The designer specifies six interfaces (the PIs) and five classes (specifications for the components): POS, GNC, PRO and two senders. Each PI provides exactly one method. For example, PI Writer_Sender includes the method public Send(). RIs are specified as attributes. For example private p:POS_Writer is an RI of Sender_1. However, no structural attributes is introduced at this level of design. For example it is not specified if the Compute operation requires a mutually-exclusive access to it.

4.2 Interface view

In the Interface view, the designer specifies all PIs with two kinds of attributes: *performance metrics*, such as deadlines, and *interaction semantics* between two methods, such as cyclic or sporadic. First, the designer draws a box to define the components, and names it, for example POS_AP (Figure 3). Then the designer imports its functional specifications from the Functional view. The signature of all methods is syntactically and semantically retained in the Interface view [10]. For example, the visibility attribute of method Write in POS has been set to public in the Functional view. The same method in POS_AP (Interface view) is *automatically* set to public. Furthermore, the designer decorates all PIs with structural attributes. For example, in GNC_AP Compute is specified by a designer as a unprotected operation and GNC_op as a cyclic

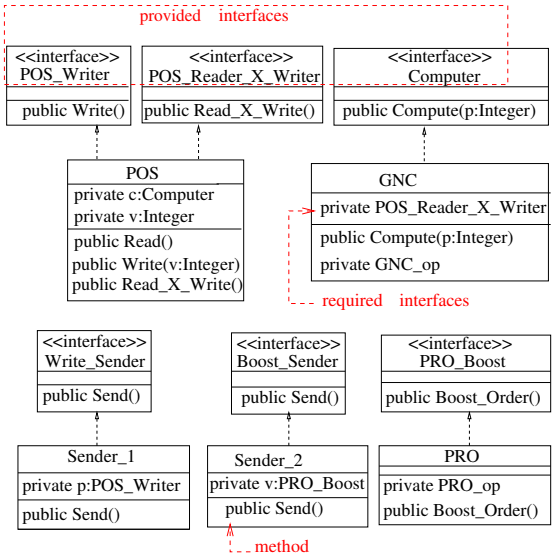

Figure 2. Functional view

operation. Finally, designer links all PI with the corresponding RIs. Figure 4 shows a screen-shot where the designer sets the attributes on performance metrics.

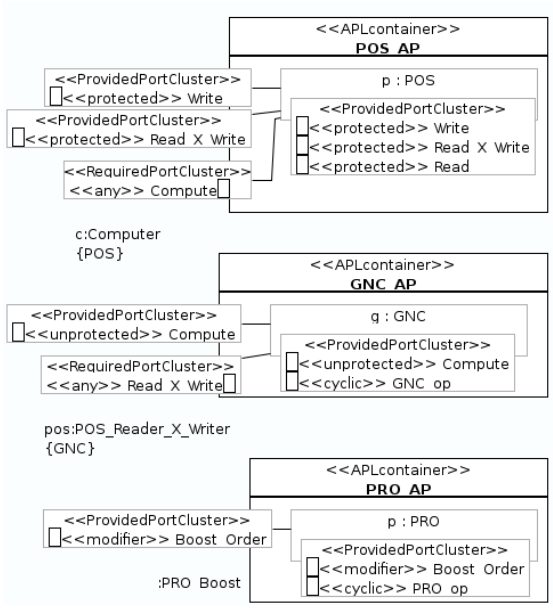

Figure 3. Interface view with attributes

4.3 Functional and Interface views

The Functional and Interface views both reside in the designer's space (PIM). One requirement of the Interface view is to preserve the information of the functional specification given in the Functional view. As a result, in the prototype the Functional view is imported in the Interface view. We have formalized this inclusion in [10] . The Functional view is characterized as a weighted directed graph HG_F in which the vertices are a finite set of PI and RI, and the edges are a finite set of their relationships. Each PI and RI has a weight specified by a combination

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

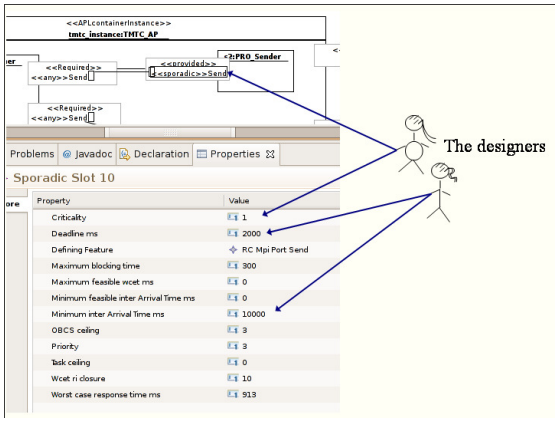

Figure 4. Attribute selection

of the functional attributes listed in tables 1 and 2. Edges are of two types: either an edge identifies a PI and its dependence on other RI, or an edge links a single RI to a single PI. The second one represents the binding of a required interface (function call) to its corresponding provided service. Thus, this is similar to a UML assembly between two RCM run-time components.

Analogously, the Interface view is characterized as a weighted directed graph HG_I which extends HG_F with the addition of the extra set of attributes and with an explicit representation of the external environment which is expected to effect the execution of the system.

By definition, graph HG_I is a fully-faithful extension of graph HG_F (by an inclusion preserving homomorphism), i.e., HG_I preserves the vertices (with their weights) and the edges of the graph HG_F. Thus, the only additional information conveyed by HG_I consists of the concurrent and temporal attributes of the vertices. (More details can be found in [10]).

Despite the inclusion, the distinction between the Functional and the Interface view is important. In particular, the same Functional view may be extended by multiple and distinct Interface views, each specifying alternative communication and synchronization architectures. Thus, in general, the relationship between the Functional and the Interface view is one to many. The designer can reuse the same (functional) diagram by simply decorating the PIs in different ways. For example, the Compute method in the GNC component can be specified in the Interface view as "protected" as well as as "unprotected". The choice reflects the intended behavior of the method at run-time (see Figure 5).

4.4 Interface and Implementation views

As discussed, cyclic is an organic and synthetic representation of a list of attributes to increase the usability of the prototype. Unlike attributes on functionality, however, attributes on the structure (PIM) depend on the run-time entities (PSM). In order to achieve correctness-by-construction, the components in the Interface view (PIM) must not only be consistent with the classes and

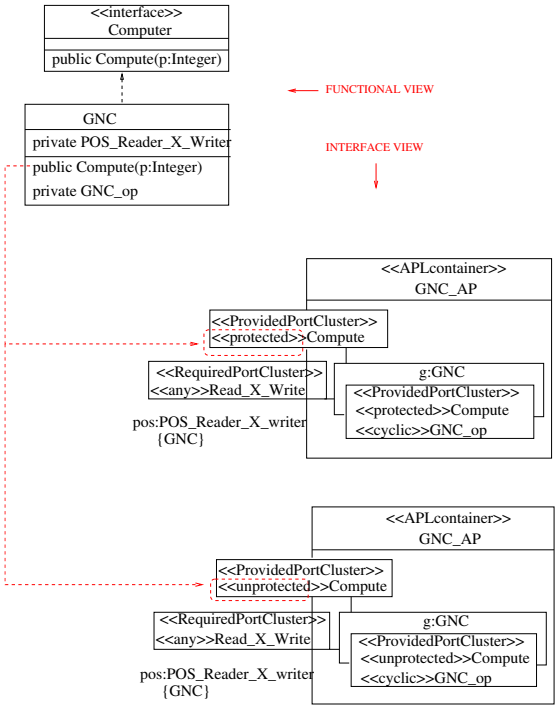

Figure 5. GNC specification with two representations in the Interface view

interfaces in the Functional view (PIM) but also with the run-time components (PSM). In the model-driven engineering approach, a run-time component is "generated" from information specified at the PIM level. To guarantee correctness-by-construction, we must have only legal run-time components, i.e., only components which are executed in the adopted platform. As a result, attributes on the structure must originate in the PSM. The main problem is therefore how to propagate information from PSM to PIM.

Our solution has been to share the same metamodel between all views so that we would have:

- consistency in the information by using the same syntax and semantics;
- propagation of information from PSM to PIM by using constraints in the metamodel.

We represent the PSM level in the Implementation view. The separation between the Functional and the Interface view must be preserved in the Implementation view. Constraints in the metamodel must propagate only information on the structure of the run-time entities. Otherwise, we could have a conflict with information imported by the Functional view. We have therefore partitioned run-time components in five constituent elements (discussed earlier in Section 3), defining the aggregation rules with respect to RCM [34], using the same metamodel and constraints on it to propagate information. Finally, we have defined atomic elements which are "pivot" of all views. Such elements are the PI.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

In Figure 6, the Functional and Interface view of the PRO component has been partnered with its representation in the Implementation view. At the bottom, the RCM run-time entity is able to capture attributes on functionality and on the structure. The inner part of the RCM run-time entity is subdivided in two parts: the first retains the methods specified in the Functional view (called OPCS, Operational Control Structure [8, 9]); the other retains the methods specified in the Interface view (the synchronization protocol, OBCS, Object Control Structure [8, 9]), and the flow of control (THREAD in the figure). OPCS, OBCS and THREAD are stereotypes in the RCM metamodel. The attributes on the structure specified in one method determine how an RCM run-time entity is aggregated. For example, the attribute cyclic determines one synchronization protocol with state dependent access and one THREAD.

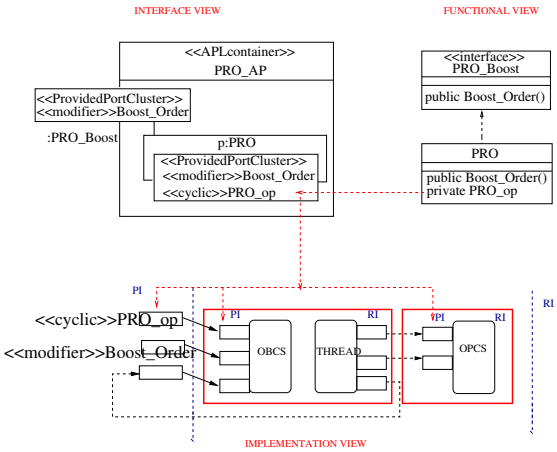

Figure 6. Functional, Interface and Implementation view

5 Industrial Assessments

The full software prototype that supports our methodology has been developed by our research team and required a total of 5.3 person/years from June 2006 to July 2007. Overall, the prototype consists of 90 metaclasses, conceptually identical to UML stereotypes, used to implement the RCM metamodel common to all modeling views; 13,000 lines of ATL to drive model transformations implemented according to the RCM Interface Grammar rules [12]; 8,000 lines of MOFscript to implement code generation; and 7,500 lines of Java to complete the graphical editor (in addition to 150,000 lines generated automatically by GMF).

Two industrial teams from the aerospace industry used and assessed the prototype independently for a total of more than 6 months in three incremental instalments. In particular, one team intensively used the prototype to redesign satellite on-board systems. Both industrial teams positively assessed the separation between functional and structural views. Their studies indicate an increase in software reuse [31, 2].

Both industrial teams also indicated a reduction of the cost in the validation and verification (V & V) phase, due to the shortening of each individual step of the software development cycle. Such a reduction is a consequence of the adopted approach, based on the model-driven engineering (MDE) together with correctness-by-construction.

The reader, interested in the study of these results, must explicitely required the authorization to the coordinator of the ASSERT Project.

6 Conclusion and Future Work

In this paper, we have discussed how the separation between attributes on the functionality and on the structure can be represented in the model-driven engineering approach in light of correctness-by-construction. This separation is based on a solid mathematical formalization. The designer specifies components in PIM by first setting the functionality of PIs and then by further decorating them with attributes on the structure. In the literature, researchers agree with using formalism like UML2 to specify attributes on functionality and with abstracting as much as possible from technology and platform. However, we must remember that in order to ensure the correctness-by-construction, attributes on the structure are not completely independent from the run-time components, which are residing in the Implementation view. This is different from what happens in the attributes on functionality. Our solution is to define two views, the Functional view and the Interface view, in which the latter includes the first. Moreover, in order to propagate the information from the Implementation view to the Interface view, we have defined only one metamodel for all views. This is how the designer can specify only correct component in the Interface view. This is why, the run-time components (PSM) must be able to preserve the separation between attributes on the functionality and on the structure.

In order to assess the viability of our methodology, our research team has realized a prototype to develop high-integrity real-time embedded systems. This prototype has been successfully tested by two teams of major European spatial industries. The main result confirms an increase of software reuse.

One of our next major challenges in the design of these systems is how to make the several different and heterogeneous formalisms coexist and interact by guaranteeing correctness-by-construction.

Acknwolegments

The research work from which this paper has originated was carried out in the ASSERT project (IST-FP6-2004 004033) partially funded by the European Commission as part of the 6^{th} Framework Programme. The views

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.

presented in this paper are however those of the authors' only and do not necessarily represent those of the other members of the ASSERT Consortium.

The authors would like to express their gratitude to all members of the META research group at the University of Padova, and the scientific supervisor T. Vardanega for having developed the prototype. In particular, we wish to thank T. Vardanega for the numerous discussions which have enhanced our understanding of the issue covered by this paper. We would also like to thank the industrial partners for testing and assessing the prototype. We thank A. Radermacher (CEA) for LaTex supporting.

References

[1] ASSERT Project. www.assert-project.net.

[2] Astrium-Toulouse (France). MPC ASSERT assessment. Technical Report 004033.MPC_ASTRIUM_SAT.DVRB.1, ASSERT - Restricted to the Consortium, 2007.

[3] F. Balarin, L. Lavagno, C. Passerone, A. L. S. Vincentelli, M. Sgroi, and Y. Watanabe. Modeling and Designing Heterogeneous Systems. In J. Cortadella and A. Yakovlev, editors, *Advances in Concurrency and System Design*. Springer-Verlag, 2002.

[4] S. Bliudze and J. Sifakis. The Algebra of Connectors - Structuring Interaction in BIP. In *Int. Conf. EMSOFT*, pages 11–20, 2007.

[5] B. Boehm and H. In. Cost vs. Quality Requirements: Conflict Analysis and Negotiation Aids. *Software Quality Professional*, 1(2):38–50, 1999.

[6] M. Bordin, M. Panunzio, and T. Vardanega. Fitting Schedulability Analysis Theory into Model-Driven Engineering. In *Euromicro SEEA*. IEEE, 2008. (to appear).

[7] R. Bruni, A. Lluch Lafuente, U. Montanari, and E. Tuosto. Style Based Reconfigurations of Software Architectures with QoS constraints. www.albertolluch.com/papers/adrqos.pdf, 2007.

[8] A. Burns, B. Dobbing, and T. Vardanega. Guide to the Use of the Ada Ravenscar Profile in High Integrity Systems. Technical Report YCS-2003-348, University of York (UK), 2003. http://www.cs.york.ac.uk/ftpdr/reports/YCS-2003-348.pdf.

[9] A. Burns and A. Wellings. *HRT-HOOD A Structural Design Method for Hard Real-Time Ada Systems*. University of York (UK), Elsevier, 1995.

[10] D. Cancila, R. Passerone, and T. Vardanega. An Algebra of the Interaction of Components in the Ravenscar Computational Model. University of Udine, Italy, 2008.

[11] D. Cancila, M. Trevisan, and T. Vardanega. (Guide for using AP-level modeling containers) HRT-UML/RCM Tutorial a gentle introduction to the HRT-UML/RCM methodology. http://www.math.unipd.it/~tullio/Research/ASSERT/Tutorial.

[12] D. Cancila and T. Vardanega. Correctness by Construction in an MDE Light. University of Udine, Italy, 2008.

[13] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource Interfaces. In *Proc. EMSOFT 2003*, 2003.

[14] R. Chapman. Correctness by Construction: A Manifesto for High-Integrity Software. http://www.stsc.hill.af.mil/CrossTalk/2005/12/0512CroxfordChapman.html, 2005.

[15] Eclipse-modeling. http://www.eclipse.org/modeling.

[16] X. Franch and P. Botella. Putting Non-Functional Requirements into Software Architecture. In 9^th *Int. Workshop on Software Specification and Design*, pages 60–67, 1998.

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

[18] J. Goodenough and L. Sha. The priority ceiling protocol: a method for minimizing the blocking of high priority ada tasks. Technical Report SEI-SSR-4, Software Engineering Institute, Pittsburgh, Pennsylvania, 1988.

[19] Software Product Evaluation - Quality Characteristics and Guidelines for their Use, 1991. ISO/IEC Standard ISO-9126.

[20] M. Kircher and M. Volter. Software Patterns, July-August 2007.

[21] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault. Mapping Specification in MDA: From Theory to Practice. In *INTEROP-ESA'05*, 2005.

[22] J. Mukerji and J. Miller. MDA Guide Version 1.0.1. www.omg.org/docs/omg/03-06-01.pdf, 2003.

[23] OMG. Catalog of Specifications. www.omgmarte.org.

[24] S. Röttger and S. Zschaler. Model-Driven Development for Non-functional Properties: Refinement through Model Transformation. citeseer.ist.psu.edu/715500.html.

[25] SAE. Architecture Analysis and Design Language. http://www.aadl.info.

[26] A. L. Sangiovanni-Vincentelli. Defining Platform-Based Design. *EEdesign*, February 2002.

[27] D. Schmidt. Model-Driven Engineering. *IEEE Computer*, pages 25–31, February 2006.

[28] B. Selic. From Model-Driven Development to Model-Driven Engineering. Keynote talk at ECRTS'07. http://feanor.sssup.it/ecrts07/keynotes/k1-selic.pdf.

[29] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Tr. on Computers*, 39(9):1175–1185, 1990.

[30] A. Snyder. Encapsulation and inheritance in object-oriented languages. In *Object-Oriented Programming Systems, Languages, and Apllications Conf. Proc.*, pages 38–45. ACM Press, 1986.

[31] Thales Alenia Space (France). HRI Experiment Report for the ASSERT Process, and the Modelling, Verification & Code Generation Tools. Technical Report 004033.HRI_TASF.DVRB.6, ASSERT - Restricted to the Consortium, 2007.

[32] L. Thiele, E. Wandeler, and N. Stoimenov. Real-Time Interfaces for Composing Real-Time Systems. In *Int. Conf. EMSOFT*, pages 34–43, 2006.

[33] T. Vardanega. A Property-Preserving Reuse-Geared Approach to Model-Driven Development. In 12^th *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 223–230. IEEE, August 2006.

[34] T. Vardanega, J. Zamorano, and J. de la Puente. On the Dynamic Semantics and the Timing Behaviour of Ravenscar Kernels. *In Real-Time Systems, Springer-Science*, 29:58–89, 2005.

[35] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison Wesley, 1998.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 27, 2009 at 10:27 from IEEE Xplore. Restrictions apply.