Component Templates for Dependable Real-Time Systems*

Matthias Tichy, Basil Becker, and Holger Giese
Software Engineering Group, University of Paderborn, Warburger Str. 100, Paderborn, Germany
[mtt|basilb|hg]@uni-paderborn.de

ABSTRACT

A general trend towards more complex technical systems can be observed which results in an increasing demand for methods and tools to develop dependable, high quality software for embedded systems. The UML in principle provides the essential concepts which are required to model such complex, safety-critical software systems. In this paper, we describe a component template plugin for the Fujaba Real-Time Tool Suite which has been especially tailored to support fault-tolerance templates such as triple modular redundancy. We report about the underlying concepts and the application of the plugin by means of an example.

1. INTRODUCTION

Due to the trend that more and more ambitious and complex technical systems are built today, an increasing demand for dependable, high quality software can be observed. This trend is characterized in [11] by very complex, highly integrated systems of systems with subsystems that must have a great degree of autonomy and, thus, are very demanding w.r.t. safety analysis. The New Railway Technology (RailCab) project[1] tackled by our efforts for the Fujaba Real-Time Tool Suite is one very extreme example for such complex systems of systems with very demanding safety requirements.

In such engineering projects, most often the involved engineers are not safety experts and, thus, sophisticated, application specific fault-tolerance techniques can often not be realized. Systematic fault-tolerance approaches such as triple modular redundancy (TMR), n-version programming, hot stand-by, etc. [14] can in contrast be employed by non experts.

However, in practice the additional complexity and pitfalls during their implementation are often a hindrance to finally achieving the intended improved dependability. We therefore propose to support the design of fault-tolerant systems by means of templates and automate the code generation for the additional logic. The templates permit to *reuse* well analyzed and understood solutions for systematic fault-tolerance and therefore minimize the risk that inadequate and error prone ad hoc solution are invented. The automatic generation of the glue logic can further exclude coding faults and therefore exclude that the additional complexity which results from the application of systematic fault-tolerance approaches themselves deteriorates the dependability of the resulting system in practice.

The UML as an object-oriented technology is one candidate to handle these safety-critical systems with software and overwhelming complexity. However, the current and forthcoming UML versions do not directly support the design of fault-tolerant designs for safety-critical system development. The presented approach tries to narrow the described gap between safety-critical system development and available UML techniques. As there is little value in proposing extensions to UML if they are not accepted by the community and tool vendors (cf. [11]), we instead propose to use only a properly defined subset of the UML 2.0 [12] component and deployment diagrams and templates for fault-tolerance to ease the task of integrating systematic fault-tolerance techniques into a UML design.

After reviewing related work in Section 2, we present our approach for component templates for fault-tolerance in Section 3. The provided tool support and application of the Fujaba Real-Time Suite Plugin are described in Section 4. We finally present some conclusions and give an outlook to planned future work.

2. RELATED WORK

Templates are a standard approach used in many different application areas. Constructs like C++ templates [15] and Java Generics [16] offer templates for programming languages. In the modeling domain templates are available in a number of different contexts.

For multimedia artifacts Cybulski [6] presents the *Template* pattern. The Template pattern is a solution to the "[...] need to produce a collection of composite artifacts similar in structure and contents." The Template pattern provides support for (1) the structural specification of a collection of composed artifacts and (2) the instantiation of the template by replacing template artifacts by concrete artifacts. Although destined for the use in multimedia applications, the Template pattern can be tailored to the use in real-time component-based applications. Our component template plugin implements an extended variant of this pattern.

The UML [12, p.541 ff.] includes a Template package. This Template packages provides Metaclasses, which allow the specification of TemplateElements which have a number of TemplateParameters. A TemplateBinding "specifies the substitution of actual parameters for the formal parameters

*This work was developed in the course of the Special Research Initiative 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

[1] http://www-nbp.upb.de

of the template". The general UML template approach allows template applications with meaningless template bindings. While such an approach is possible for modeling standard software, we target the domain of safety-critical embedded systems. Thus, a more strict template mechanism is required. The presented template mechanism is similar to the general one in the UML, but includes special support for real-time component templates in contrast to the general UML template mechanism. Here special support means, that e.g. if a component has been used in the application of a template, only ports of this component and not arbitrary ones can be used.

3. COMPONENT TEMPLATES

In the following, we will present the Fujaba real-time component diagrams. Thereafter, we will show how component templates are specified and how they can be applied.

The Fujaba real-time component diagrams [5, 8] are based on concepts originally proposed in ROOM and UML 2.0 [12, 13]. They are used for the specification of system structure. Component diagrams specify components and their interaction in form of connectors. We differentiate component types and their instances during runtime. Connectors model the communication between different components via ports and interfaces. Ports are distinct interaction points between components and are typed by provided and required interfaces. Behavior of components is specified by real-time statecharts [3]. As this paper deals with structural templates, we will, in the following, not consider the behavioral aspect of components and component templates. As example, we will use the triple-modular-redundancy (TMR) fault tolerance technique.

3.1 Template specification

In a first step, the structure of the fault-tolerance technique must be specified. This is done by creating a component template specification. This specification is merely a standard real-time component diagram.

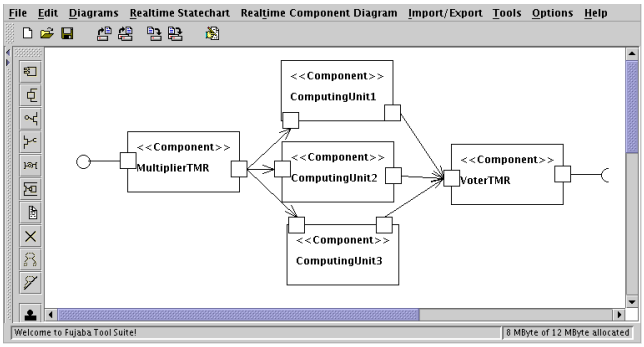

Figure 1: Component template specification for TMR

Figure 1 shows the component template specification for the triple modular redundancy fault tolerance technique. A triple modular redundancy system uses a multiplier component which triples the input received and forwards it to the three services ComputingUnit1 ... 3, which actually perform the computation. The voter compares the different results and chooses the result which at least two of the components returned. Thus, a triple modular redundancy system can tolerate one crashed or malfunctioning service. The ports and interfaces attached to the left of the MultiplierTMR component and right of the VoterTMR component are not part of the fault tolerance pattern but are important for connecting the using and used components during application of the component template.

3.2 Application

The specified fault tolerance component templates are later used to build more robust applications by applying them to the component structure of a system. First, an appropriate component template is selected and added to the component structure. Then, the different parts of the component template must be replaced by the actual implemented parts. The different parts here are components, ports, and interfaces. Thus, a mapping must be defined by the user between the components of the component template and implemented components. Thereafter, the ports of this template component and the ports of the implemented component are mapped. Finally, the interfaces attached to the ports are mapped. The mapping of interfaces must respect the type of the interfaces, e.g. *required* interfaces of the template component must be mapped to *required* interfaces of the implemented component. In addition the interface of the implemented component must be a subclass of the interface of the template component. This constraint offers support for more application specific templates where consistency between interface types is required. However, this constraint can be relaxed for broader usage of the component template.

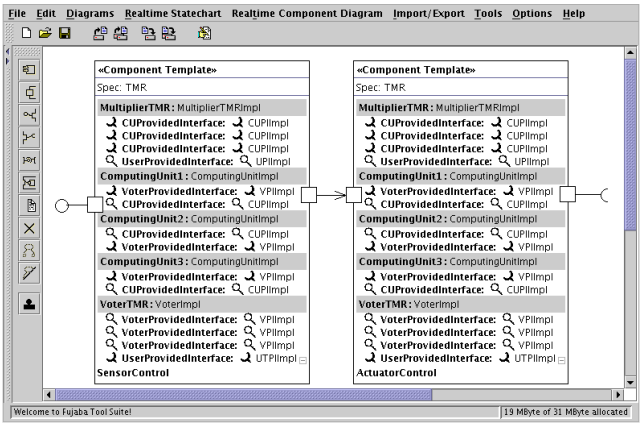

Figure 2: Application of the TMR template

Figure 2 shows the application of the TMR component template. The TMR template is applied for the sensor control software as well as for the actuator control software as shown on the bottom of each component in Figure 2. The different parts of the TMR template are already mapped to implemented components, ports, and interfaces.

After all parts of the template are mapped to implemented parts, the template in the component diagram is replaced by the template structure (cf. Figure 1) using the mapped implemented parts. This replacement, thus, makes the fault tolerance techniques explicit in the design of the system. After this step, the standard code generation of the Fujaba Real-Time Tool Suite can be employed to synthesize the source code for the fault-tolerance enhanced system.

3.3 Multistage arrangement

A fault tolerance template like TMR can be employed multiple times in the component structure of an embedded system. A naive usage of this template would result in a situation where three redundant components are connected to three other redundant components by single voter and multiplier components. Thus, the redundancy gained by the application of the TMR template is defeated by the single-point-of-failure voter and multiplier components.

A better approach is the usage of a multistage arrangement. A multistage arrangement uses redundant voter and multiplier components in contrast to the mentioned single voter and multiplier approaches. Thus, a transformation of multiple applications of TMR or other templates to a multistage arrangement is important for the fault-tolerance of the system. Using the Story-Pattern language [7, 10] of Fujaba, it is possible to specify an accordant transformation from a multiple application of TMR to a multistage arrangement.

After this description of component templates, we present in the next section the tool support offered by the component template plugin.

4. TOOL SUPPORT

We have developed a Fujaba plugin that provides tool support for the mentioned component template specifications and the mappings between template components, ports and interfaces and implemented ones. In the following, we highlight the plugin dependencies, the meta-model extension, and special mapping support.

Plugin structure

The developed RealtimeComponentTemplate plugin (RCT plugin) depends on two other plugins developed at the University of Paderborn. First it depends on the RealtimeComponent (cf. [5]) plugin. We use the RealtimeComponent plugin's component diagrams for the specification of component templates. As the RealtimeComponent plugin depends on the RealtimeStatechart plugin [3] our RCT plugin depends on it, too.

Meta-model extension

How are the mappings between components, interfaces and ports realized? At first sight one might think that this won't be a problem, because all used classes are defined in one plugin - the RealtimeComponent plugin. The easiest way would be to define one-to-many self-associations between the component, port, and interface classes. But then we would have to change the RealtimeComponent plugin which we wanted to avoid. Fortunately, Fujaba provides the ASG mechanism [4] to avoid these problems. In the following paragraphs, we will restrict our explanations to the mapping of components. The mapping of ports and interfaces is done analogously.

Each mapping between components is represented by an instance of RTCompMapping. This mapping has two references to an extension of ASGElementRef; one for the incoming and one for the outgoing mapping. These extensions of ASGElementRef have a one-to-one ASG reference to the Component class. As we need one-to-many associations between Component and RTCompMapping (e.g. one component from a given specification is used in more than one template application), however ASG only provides one-to-one associations, we implemented the one-to-many association between RTCompMapping and the ASGElementRef extensions. The ASGElementRef extensions have the role of a proxy for the Component class. This means we displaced the one-to-many association needed in our plugin. As mentioned earlier the interface and port mapping is implemented in the same way.

A few words on the hierarchy of the different mapping types: Each template has a set of component mappings to implemented components. Since every port is owned by exactly one component, the port mappings are stored in each component mapping. The same argumentation applies for interface mappings.

Mapping support

In the following, we present the way in which the plugin supports the mapping specification. Tool usability[2] is a major factor for the acceptance of an approach and its supporting tool. Therefore, we explicitly emphasized usability especially for the mapping specification.

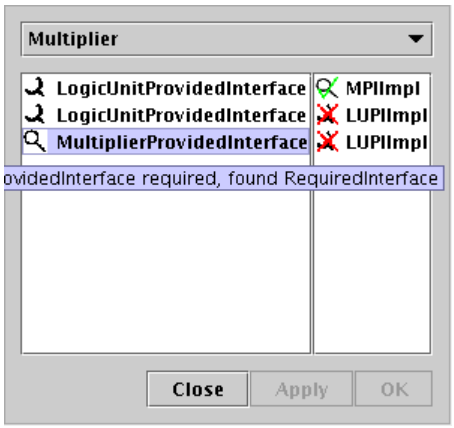

Figure 3: Interface mapping dialog

As mentioned, interface mapping is subject to constraints regarding the type of the interfaces (required/provided) and the subclass relation between template interfaces and implemented interfaces. To improve the usability of this interface mapping the interface mapping dialog produces immediate, easy to understand feedback. This means every time you select two interfaces you want to map, the plugin checks whether this mapping will be correct or not. A mapping of interfaces is correct if both interfaces are of the same type (provided or required interface) and the interface declared in the implementation is a subclass of the interface in the specification. This direct feedback is realized either by a red crossed out or by a green checked off interface icon (cf. Figure 3). If the mapping is incorrect the tooltip of the interface in the implementation gives a short explanation why the mapping is wrong.

[2]To introduce our understanding of usability we will first give a short definition of it found in [1]: "Usability is the ease with which a user can learn to operate, prepare inputs for and interpret outputs of a system or component."
The expression usability is used in different contexts. In our context, i.e. software development, usability is often called *software ergonomics*. Ergonomics is the conformity of technology (i.e. software) to human psychophysical capabilities. For further details on ergonomic and usable software see [2].

Concerning the order of the different mappings, the standard order would be mapping components, then ports, and then interfaces. We provide a small shortcut for this mapping based on the fact that a interface can only be connected to exactly one port, i.e. if you know the interface, you know the corresponding port. The plugin exploits this property and allows you to map the interfaces without mapping the ports. Every time you map an interface, the corresponding port mapping is automatically determined.

5. CONCLUSION AND FUTURE WORK

We presented an approach for specification of dependable, component-based, embedded systems. We aim for improving the fault tolerance of distributed systems by the application of fault tolerance component templates. Tool support for specification and application of component templates has been developed including special care for the usability of the mapping.

Currently, the component templates only cover the structural parts of fault tolerance techniques. In the future, we will also consider the glue logic resulting of the behavior of voter and multiplier components. We believe, that the behavior of these two component types can in principle be synthesized based on the behavior of the other components.

In the future, we will consider the deployment issues of fault tolerance templates. Fault tolerance templates typically employ redundancy to enhance fault tolerance. If the redundant components of a fault tolerance template are deployed to the same host, there is no fault tolerance w.r.t. failures of this host. Thus, appropriate deployment constraints must be specified for fault tolerance templates. These constraints would specify that each of the redundant components must be deployed to different hosts.

In [9], an approach for a compositional hazard analysis of component-based systems is presented. The knowledge about the employed fault tolerance provided by the templates could be exploited in the hazard analysis.

Currently, the user can use arbitrary components in the application of the fault tolerance template. Fault tolerance techniques like n-version programming explicitly request heterogeneous components in order to tolerate systematic implementation errors. Adding appropriate constraints to component templates would ease the use of component templates for heterogeneous fault tolerance techniques.

REFERENCES

[1] *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers, New York, 1990.

[2] A. Brenneke, R. Keil-Slawik, and W. Roth. Designorientierung und Designpraxis - Entwicklung und Einsatz von konstruktiven Gestaltungskriterien. In U. Arend, E. Eberleh, and K. Pitschke, editors, *Software-Ergonomie '99 Design von Informationswelten*, pages 43–52. B. G. Teubner Stuttgart, 1999.

[3] S. Burmester and H. Giese. The Fujaba Real-Time Statechart PlugIn. In *Proc. of the Fujaba Days 2003, Kassel, Germany*, October 2003.

[4] S. Burmester, H. Giese, J. Niere, M. Tichy, J. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool Integration at the Meta-Model Level within the FUJABA Tool Suite. *International Journal on Software Tools for Technology Transfer (STTT)*, 2004. (accepted).

[5] S. Burmester, M. Tichy, and H. Giese. Modeling Reconfigurable Mechatronic Systems with Mechatronic UML. In *Proc. of Model Driven Architecture: Foundations and Applications (MDAFA 2004), Linköping, Sweden*, June 2004.

[6] J. L. Cybulski and T. Linden. Composing Multimedia Artifacts for Reuse. In *Proc. of the 1998 Pattern Languages of Programs Conference, Monticello, Illinois, USA*, August 1998.

[7] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language. In G. Engels and G. Rozenberg, editors, *Proc. of the* 6^{th} *International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, LNCS 1764, pages 296–309. Springer Verlag, November 1998.

[8] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland*, pages 38–47. ACM Press, September 2003.

[9] H. Giese, M. Tichy, and D. Schilling. Compositional Hazard Analysis of UML Components and Deployment Models. In *Proc. of the 23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP), Potsdam, Germany*, Lecture Notes in Computer Science. Springer Verlag, September 2004. (to appear).

[10] H. Köhler, U. Nickel, J. Niere, and A. Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. of the* 22^{nd} *International Conference on Software Engineering (ICSE), Limerick, Ireland*, pages 241–251. ACM Press, 2000.

[11] J. A. McDermid. Trends in Systems Safety: A European View? In P. Lindsay, editor, *Seventh Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, volume 15 of *Conferences in Research and Practice in Information Technology*, pages 3–8, Adelaide, Australia, 2003. ACS.

[12] Object Management Group. *UML 2.0 Superstructure Specification*, 2003. Document ptc/03-08-02.

[13] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons, Inc., 1994.

[14] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.

[15] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1991. Second Edition.

[16] M. Torgersen, C. P. Hansen, E. Ernst, P. von der Ahé, G. Bracha, and N. Gafter. Adding wildcards to the java programming language. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1289–1296. ACM Press, 2004.