Evaluation of a Tool for Supporting Software Component Services in Embedded Real-Time Systems

Frank Lüders and Ivica Crnkovic
Mälardalen University
Dept. of Computer Science and Electronics
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 {15 17 28, 10 31 83}
{frank.luders, ivica.crnkovic}@mdh.se

Per Runeson
Lund University
Dept. of Communication Systems
PO Box 118, SE-221 00 Lund, Sweden
+46 46 222 93 25
per.runeson@telecom.lth.se

ABSTRACT
The use of software component models has become popular in the development of desktop applications and distributed information systems. The most successful models incorporate support for run-time services of general use in their intended application domains. There has been no widespread use of such models in the development of embedded real-time systems and much research is currently directed at defining new component models for this domain. We have explored the alternative approach of extending a mainstream component model with run-time services for embedded real-time systems. A prototype tool has been developed that generates code for a number of such services. To evaluate this tool, we have conducted a multiple-case study, where four teams of students were given the same development task. Two teams were given the tool while the remaining two were not. This paper describes the design of the study and our initial analysis of the results.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *modules and interfaces, object-oriented design methods, Microsoft Windows*.

D.2.9 [**Software Engineering**]: Management – *productivity*.

C.3 [**Special-Purpose and Application-Based Systems**]: *real-time and embedded systems.*

General Terms
Design, Economics, Experimentation, Human Factors.

Keywords
Software Component Services, Microsoft Component Object Model.

SERPS'06, October 18-19, 2006, Umeå, Sweden.

1. INTRODUCTION
The use of software component models has become popular over the last decade, in particular in the development of desktop applications and distributed information systems. The most successful component models in these domains include JavaBeans [1] and ActiveX [2] for desktop applications and Enterprise JavaBeans (EJB) [3] and COM+ [4] for information systems. In addition to basic standards for naming, interfacing, binding, etc., these models also define standardized sets of run-time services oriented towards the application domains they target. This concept is generally termed software component services [5].

Software component models have not been widely used in the development of real-time and embedded systems. It is generally assumed that this is due to the special requirements such systems have to meet, in particular with respect to timing predictability and limited use of resources. Much research has been directed towards defining new component models for real-time and embedded systems, typically focusing on relatively small and statically configured systems. Most of the published research proposes models based on source code components and targeting relatively narrow application domains. Examples of such models include Koala for consumer electronics [6] and SaveCCM for vehicle control systems [7].

An alternative approach is to strive for a component model for embedded real-time systems based on binary components and targeting a broader domain of applications, similarly to the domain targeted by a typical real-time operating system. In our previous work we have explored the possibility of using a mainstream component model as the starting point and extending it with software component services for embedded real-time systems [8]. Specifically, we have investigated the use of the Component Object Model (COM) [9] with the real-time operating system Windows CE [10] and developed a prototype tool that generates code for a number of services.

To evaluate the usefulness of the prototype tool, we have conducted a multiple-case study where four development projects were run in parallel. Two of these used the tool and two did not. Before describing the study, we briefly present the prototype tool and its rationale in Section 2. Section 3 describes the design of the case study, Section 4 discusses the process of data collection in more detail, and Section 5 presents our initial analysis of the results. Some related work is briefly reviewed in Section 6 while Section 7 presents conclusions and our plans for future work.

2. BACKGROUND

Software component models like EJB and COM+ include support for various services that are generally useful in the domain of distributed information systems. Examples of such services include transaction control, data persistence, and security. Our focus here is on services that address common challenges in embedded real-time systems, including logging, synchronization, and timing control. Although the sets of services are different, the principles used to provide the run-time services are similar in many respects.

A prototype tool for supporting software component services in embedded real time systems was presented in [8]. The tool adds services to COM components on Windows CE through the use of proxy objects that intercept method calls. Figure 1 illustrates the use of a proxy object that provides a simple logging service. The object C2 implements an interface IC2 for which one wishes to apply a logging service. A proxy object that also implements IC2 is placed between C2 and a client that uses the operations exposed through IC2. The operations implemented by the proxy forward all invocations to the corresponding operations in C2 in addition to writing information about each invocation to some logging medium.

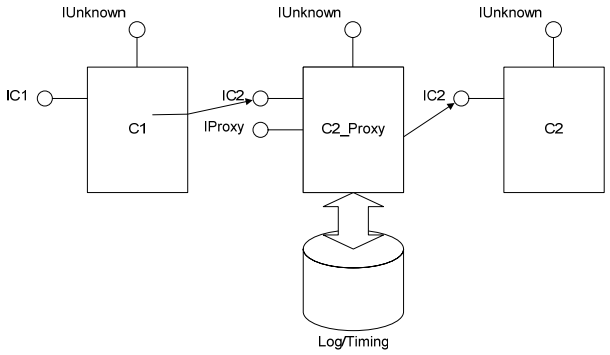

Figure 1. A logging service proxy.

The tool takes as inputs a component specification along with specifications of desired services and generates source code for a proxy object. Component specifications may be in the form of Interface Definition Language (IDL) files or their binary equivalent Type Library (TLB) files. Desired services are either specified in a separate file using an XML-based format or in the tool's graphical user interface, described further below. Access to component source code is not required. Based on these inputs, the tool generates a complete set of files that can be used with Microsoft eMbedded Visual C++ to build a COM component implementing the proxy objects (i.e., the proxies are themselves COM objects). This process is depicted in Figure 2.

This use of proxy objects for interception is inspired by COM+. However, rather than to generate proxies at run-time, they are generated and compiled on a host computer and downloaded to the embedded system along with the application components. This process may occur when the software is initially downloaded to the system or as part of dynamic reconfiguration of a system that supports this. In the latter case, one can imagine updating or adding proxies without updating or adding any application components. The current version of the tool only generates proxy code and does not address the registration and run-time instantiation of components. This means that the client code must instantiate each proxy along with the affected COM object and set up the necessary connection between them.

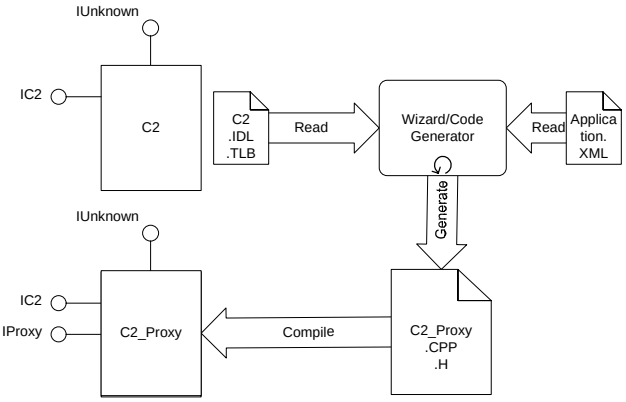

Figure 2. Proxy object generation.

Figure 3 shows the graphical user interface of the tool. After a TLB or IDL file has been loaded all COM classes defined in the file are listed. Checking the box to the left of a COM class causes a proxy for that class to be generated when the button at the bottom of the tool is pressed. Under each COM class, the interfaces implemented by the class is listed and, under each interface, the operations implemented by the interface. In addition, the available services are listed with their names set in brackets. Checking the box to the left of a service causes code to be generated that provides the service for the element under which the service is listed. In the current version of the tool, a service for cyclic execution may only be specified for the IPassiveController interface while all other services may only be specified for individual operations. The IPassiveController interface is described further below.

Checking the logging service results in a proxy that logs each invocation of the affected operations. The timing service causes the proxy to measure the execution time of the operation and write it to the log at each invocation (if timing is checked but not logging, execution times will be measured but not saved). The synchronization service means that each invocation of the operation will be synchronized with all other invocations of all other operations on the proxy object for which the synchronization service is checked. The only synchronization policy currently supported is mutual exclusion.

The timeout service has a numeric parameter. When this service is selected (by clicking the name rather than the box) as in Figure 3, an input field marked Milliseconds is visible near the bottom of the tool. Checking the service results in a proxy where invocations of the operation always terminate within the specified number of milliseconds. In the case that the object behind the proxy does not complete the execution of the operation within this time, the proxy forcefully terminates the execution and returns en error code.

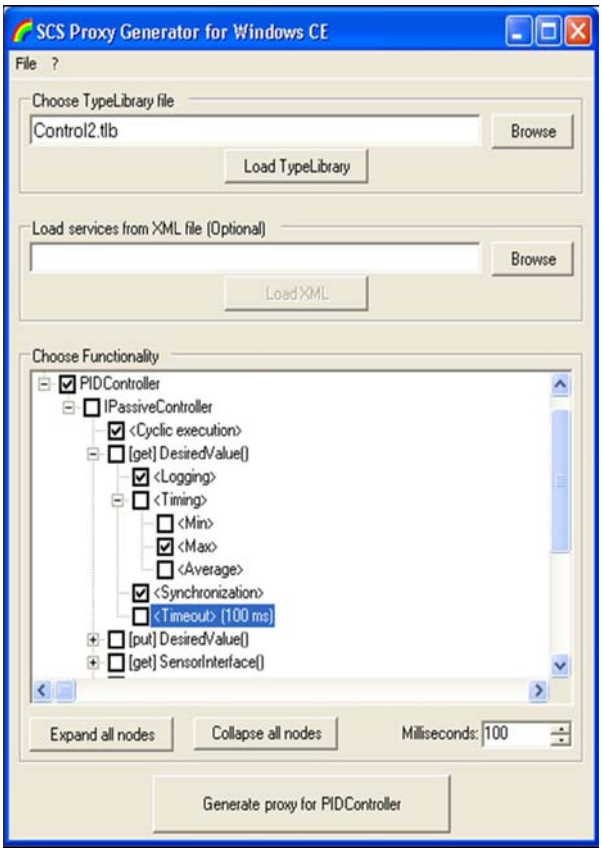

Figure 3. User interface of the prototype tool.

The cyclic execution service is particularly suited for components that implement process controllers [11]. If this service is checked, the proxy will implement an interface called IActiveController instead of IPassiveController. Both interfaces share a common set of operations for accessing control parameters, including the controller's set point. IActiveController includes operations for setting the period and threading priority of the cyclic execution. IPassiveController includes one operation for updating the controller's output and one for updating the its internal state. The proxy invokes both these operations cyclically and the latter is synchronized with the operations for accessing control parameters.

3. CASE STUDY DESIGN

In order to evaluate the tool support, we launched an empirical study. The study is conducted using a multiple-case study design [12]. We prefer considering it a case study rather than an experiment, since from an experimental point of view, it is a quasi-experimental "post-test non-equivalent groups design" according to Robson's terminology [13, pp. 133-146]. We observe four different project teams, solving the same problem with two different sets of working conditions – access to tool or not. We measure their results in quantitative terms of time consumption, problem reporting and a qualitative analysis of their technical solutions. We can not distinguish quantitatively between the effects of the tool and the teams' capabilities, but seen as a case study, we may find indications and opinions regarding the value and contribution of the tool.

The study was conducted in the context of a project assignment for third year students in computer science that runs over 10 weeks with 50% workload –corresponding to 7.5 standard European credit units. There were 30 students, who were divided into four project teams of seven or eight members. During the early phases of the projects, some students dropped of from the course, such that the team sizes varied from five to eight members.

The assignment of the projects was to develop a component-based application to be run under Windows CE on a PC connected to two water tanks where the water level can be controlled by individual pumps. A requirement was that the software should include a component implementing a PID controller [11] able to control the pumps. The controller had to sample the current water level and update the pump voltage in a timely fashion. It should furthermore be possible to change the desired water level and control parameters during the operation of the controller in a thread-safe manner.

The detailed requirements of each project were elicited by the project teams through negotiation with a course instructor acting as customer. Thus, the requirements were not identical. Over the course of the projects, some changes in the requirements were introduced by the customer. This was in part based on each team's achievements to avoid the task being too simple for some teams. In addition, two of the teams were given the additional requirement that they should use the prototype tool to implement multithreading, synchronization, and logging of process data,

The design used for the study is summarized as follows:

1. The subjects were divided into four teams by the course instructors with the intention of making the teams as equal as possible.
2. The team capabilities were assessed based on the earliest phases of the projects – requirements capture and user interface prototyping. We found that two teams were "strong" and two teams were "weak".
3. All four teams were given (almost) the same task – implementation of the control system for a water tank. One strong and one weak team were given access to the tool, while the other two teams were not.
4. Data was collected during the course of the project from time sheets and weekly project reports, and the project deliverables were assessed – a project description, project final report, design description and code.

The case study teams are summarized in Table 1.

Table 1. Case study design overview.

	Tool support	No tool support
Strong team	Team 1	Team 2
Weak team	Team 3	Team 4

Threats to the validity of a case study may be grouped into three categories; reactivity, respondent bias and researcher bias [13, p.172]. Reactivity means that the studied phenomenon behaves

differently due to the fact that it is observed. The studied context is clearly artificial and observed in a teaching context, but all four teams are observed in the same way. Respondent bias means the risk that the respondents act based on expectations. The tool evaluation is a minor part of the study, and hence it is not clear to them what is expected. Further, the triangulation using both quantitative and qualitative measurements reduces the bias. Researcher bias means the risk that the researchers only see the positive signs pro their proposed tool. This is addressed by involving a third author for peer debriefing and negative case analysis. Triangulation also reduces researcher bias.

4. DATA COLLECTION

Each of the four project teams were charged with delivering a number of documents during the course of the project. In addition, the status of each project was presented orally at weekly meetings with a steering group, consisting of two course instructors for each project. Among the information collected was the number of working hours for each team member and activity. Table 2 summarizes the reported working hours per activity for each group in number of hours as well as in percent of the total.

Table 2. Reported working hours.

Activity	Team 1	Team 2	Team 3	Team 4
Project management	80 h 6%	37 h 3%	131 h 12%	120 h 17%
Configuration management	40 h 3%	23 h 2%	64 h 6%	40 h 6%
Requirements management	400 h 28%	210 h 20%	78 h 7%	70 h 10%
Software design	280 h 19%	160 h 15%	131 h 12%	80 h 11%
Software coding	480 h 33%	345 h 32%	290 h 26%	220 h 31%
Software testing	160 h 11%	160 h 15%	115 h 10%	131 h 18%
Other activities	0 h 0%	130 h 12%	300 h 27%	60 h 8%
Total	1440 h 100 %	1065 h 100 %	1109 h 100 %	721 h 100 %

Obviously, each team was also expected to deliver a number of software components. At the end of the project, the executable software was demonstrated with the target equipment and all components – including source code – were delivered. Since the tool under evaluation is primarily intended to help with the implementation of cyclic execution and synchronization, we inspected the source code of all teams with respect to thread safety and timeliness. More specifically we studied the controller component and its relation to other components to determine if the following criteria were met:

- A timing mechanism is used to ensure that the control loop executes with the correct cycle time.
- A synchronization mechanism is used to prevent set-points and control parameters from being written by the application while they are being read by the control loop.

The properties of each team's controller component are summarized in Table 3 and described in more detail below.

Team 1 had used the prototype tool and the cyclic execution service to generate a proxy that ensured correct timing of the control loop. Team 3 also used the tool to generate a proxy for the controller, but had failed to select the cyclic execution. Instead they had manually written code to execute the control loop in a separate thread, as had Teams 2 and 4, who did not use the tool at all. These three teams had all used appropriate timing mechanisms correctly.

Table 3. Control loop properties.

	Team 1	Team 2	Team 3	Team 4
Timely	Yes	Yes	Yes	Yes
Thread safe	No	No	No	No

Although Team 1 had used the tool with the cyclic execution service, they had failed to ensure thread safe execution of the control loop. As described in Section 2, the interface IPassiveController contains one operation for updating the controller output and one for updating its internal state, and only the latter is synchronized with other operations. Team 1's component was not thread safe, because the first operation updates the output as well as the internal state, while the implementation of the latter operation was left empty.

Of the remaining three teams, who had not used the cyclic execution service, Teams 2 and 3 had not used any synchronization mechanism at all in neither the control loop nor the operations for accessing the controller's data. Team 4 had used a mutual exclusion mechanism in the control loop but not in the other operations; the mechanism had been used in such a way that the control loops for the two water tanks were (quite unnecessarily) synchronized with each other. Consequently, none of these controller components are thread safe either.

5. ANALYSIS

Based on the collected data, described in the previous section, we have performed a preliminary analysis to see whether there are any indications that the different conditions for the four project teams – i.e. use of tool or not – has resulted in any significant differences in the projects' results. The analysis is somewhat complicated by the fact that Team 3, who used the tool, failed to use the cyclic execution service. Thus, with respect to implementation of the cyclic execution of the control loop, this team should be considered as not having used the tool, as indicated in Table 4.

Table 4. Overview of teams with respect to cyclic execution of the control loop.

	Tool support	No tool support
Strong team	Team 1	Team 2
Weak team		Teams 3 and 4

The reported worked hours for the four teams, summarized in Table 1, reveals no correlation between the use of the tool and the

number of worked hours for the different activities. This is true both for the absolute number of hours as well as the percentages of the worked hours spent on the different activities. In particular, there are no significant differences with respect to the relative amount of work required for software coding. This can probably be attributed, at least in part, to the fact that the amount of code generated by the tool constitutes relatively small portions of the total amount of the code produced by the projects. Thus, a more detailed investigation of the working hours related to those parts of the software where the tool is most effective – i.e. the implementation of the control loop with multithreading and synchronization – would be desirable.

The properties of the four teams' controller components summarized in Table 3 shows a success rate of zero when it comes to thread safe execution of the control loops. Before analyzing this further, it should be pointed out that the subjects did not have prior knowledge of neither real-time systems in general nor computer-based control systems in particular. Although the necessity of using some synchronization mechanism to ensure thread safety was pointed out by the instructors at the start of the project, it seams that this was not made sufficiently clear, as at least two of the teams completely neglected to address the issue. This is not an unexpected mistake from someone without experience in concurrent system development, in particular since the error only occasionally results in failure and is likely to go undetected by testing.

Of the two teams whose control loops included some synchronization mechanism Team 1 had used the tool to generate the synchronization code. The fact that the team had only implemented the operation intended to update the controller output prior to synchronization may be an indication that they too did not realize the need for synchronization, although an alternative scenario is that they were mistaken and believed that synchronization was provided. In any case, this observation shows that the way we have chosen to implement the tool to rely on two operations for updating the output and internal state respectively, is a potential source of error. This potential could easily be eliminated at the cost of removing the ability to generate the controller output in a way that is guarantied not to be blocked by threads of lower priority. Another possible improvement may be to rename the operations from UpdateOutput and UpdateState to reflect that the former operation do not support thread safety.

Team 4 seems to have attempted to ensure thread safe execution of the control loop by using a mutual exclusion mechanism. The attempt failed because other operations that may update the controller's state did not use the same mechanism. A possible interpretation of this observation is that the team erroneously assumed that using the mechanism, called critical section in Windows CE, would prevent the thread executing the control loop from conflicting with any other threads in the system.

6. RELATED WORK

The major source of inspiration for our approach and the prototype tool presented in this paper is COM+ [4], which is Microsoft's extension of their own COM model with services for distributed information systems. These services provide functionality such as transaction handling and persistent data management, which is common for applications in this domain and which is often time consuming and error prone to implement for each component. We use the same criteria for selecting which services our component model should standardize, namely that they should provide non-trivial functionality that is commonly required in the application domain. Since our component model targets a different domain than COM+, the services we have selected are different from those of COM+ as well.

We are furthermore inspired by the technique of providing services by interception. This mechanism is also used in other technologies and is sometimes called interceptors rather than proxies, e.g. in the Common Object Request Broker Architecture (CORBA) [14] and the MEAD framework for fault-tolerant real-time CORBA applications [15].

The approach presented in this paper is similar to the concept of aspects and weaving. The real-time component model RTCOM [16] supports weaving of functionality into components as aspects while maintaining real-time policies, e.g. execution times. However, RTCOM is a proprietary source code component model. Moreover, functionality is weaved in at the level of source code in RTCOM whereas in our approach, services are introduced at the system composition level.

Another effort to support binary software components for embedded real-time systems is the ROBOCUP project [17], which builds on the aforementioned Koala model and primarily targets the consumer electronics domain. This work is similar to ours in that the component model defined as part of this project is largely based on the basic concepts of COM. Furthermore, the sequel of the project, called Space4U [18], also seems to use a mechanism similar to proxy objects, e.g. to support fault-tolerance.Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

7. CONCLUSION AND FUTURE WORK

This paper describes a multiple-case study we have launched to evaluate the usefulness of a prototype tool that supports the concept of software component services in embedded real-time systems. The study is based on four parallel software development projects, where two of the project teams were given the tool. One of these only partly used the tool as intended, however, so in some important respects, three of the projects were conducted without tool support and only one with tool support.

The projects are completed and have resulted in delivery of documentation and software from each of the four teams. This paper presents our first analyses of some of this data – the reported number of working hours for different activities and the properties of the delivered software with respect to timeliness and thread safety. We have not been able to draw any conclusion from the reported working hours, except that it is desirable to study the required development effort related to certain parts of the software in more detail.

The analysis of software properties has shown that the students participating in the projects were not well prepared for implementing the required functionality in a thread safe manner, neither with the support of the tool nor without it. However, we have identified possible changes to the tool that would probably make it easier to avoid such errors, even for developers without experience of multithreaded software.

In the immediate continuation of the work presented here we plan to expand upon our analysis of the differences between the four

projects. In addition to the already identified task of analyzing the development effort in more detail, we expect to undertake a more comprehensive and systematic qualitative analysis of the delivered documentation and software.

We also plan to launch further empirical studies to evaluate our approach for software component services and the prototype tool. For instance, it would be of great interest to investigate the use of the tool in connection with reuse of components across projects. One possibility is to conduct another study with students as participants, either as a multiple-case study again or as a controlled experiment.

It would also be desirable to apply the prototype tool in an industrial case study, which would imply a lower level of replication and control but allow us to evaluate our approach in a more realistic setting. We plan to evaluate and extend the set of services supported by the tool. We hope to do this with the help of industrial partners in such domains as industrial automation, telecommunications, and vehicle control systems.

8. REFERENCES

[1] R. Englander, *Developing Java Beans*, O'Reilly, Sebastopol, CA, USA, 1997.

[2] D. Chappell, *Understanding ActiveX and OLE*, Microsoft Press, Redmond, WA, USA, 1996.

[3] R. Monson-Haefel, B. Burke, and S. Labourey, *Enterprise JavaBeans*, 4th edition, O'Reilly, Sebastopol, CA, USA, 2004.

[4] D.S. Platt, *Understanding COM+*, Microsoft Press, Redmond, WA, USA, 1999.

[5] G.T. Heineman and W.T. Council, *Component-Based Software Engineering – Putting the Pieces Together*, Addison-Wesley, Reading, MA, USA, 2001.

[6] R. van Ommering, F. van der Linden, and J. Kramer, "The Koala Component Model for Consumer Electronics Software", *Computer*, volume 33, issue 3, March 2000, pp. 78-85.

[7] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren, "SaveCCM – A Component Model for Safety-Critical Real-Time Systems", *Proc. 30th EROMICRO Conference*, Rennes, France, 2004, pp. 627-635.

[8] F. Lüders, D. Flemström, I. Crnkovic, and A. Wall, "A Prototype Tool for Software Component Services in Embedded Real-Time Systems", *Proc. 9th International Symposium on Component-Based Software Engineering*, Västerås, Sweden, 2006, pp. 222-237.

[9] D. Box, *Essential COM*, Addison-Wesley, Reading, MA, USA, 1997.

[10] J. Murray, *Inside Microsoft Windows CE*, Microsoft Press, Redmond, WA, USA, 1998.

[11] K.J. Åström and B. Wittenmark, *Computer Controlled Systems – Theory and Design*, 2nd edition, Prentice Hall, Englewood Cliffs, NJ, USA, 1990.

[12] R.K. Yin, *Case Study Research – Design and Methods*, 3rd edition, Sage Publications, Thousand Oaks, CA, USA, 2003.

[13] C. Robson, *Real World Research*, 2nd edition, Blackwell Publishing, Oxford, UK, 2002.

[14] Object Management Group, "Common Object Request Broker Architecture – Core Specification", OMG formal/04-03-12, March 2004.

[15] P. Narasimhan, T.A. Dumitras, A.M. Paulos, S.M. Pertet, C.F. Reverte, J.G. Slember, and D. Srivastava, "MEAD – Support for Real-Time Fault-Tolerant CORBA", *Concurrency and Computation – Practice and Experience*, volume 17, number 12, February 2005, pp. 1527-1545.

[16] A. Tesanovic, D. Nyström, J. Hansson, and C. Norström, "Aspects and Components in Real-Time System Development – Towards Reconfigurable and Reusable Software", *Journal of Embedded Computing*, volume 1, number 1, February 2004, pp. 1-16.

[17] J. Muskens, M.R.V. Chaudron, and J.J. Lukkien, "A Component Framework for Consumer Electronics Middleware", C. Atkinson et al. (eds.), *Component-Based Software Development for Embedded Systems*, Springer, Heidelberg, Germany, 2005, pp. 164-184.

[18] Space4U Project, "Space4U Public Home Page", http://www.hitech-projects.com/euprojects/space4u/, January 2006 (accessed 28 April 2006).