Model Driven Code Generation for Critical and Adaptative Embedded Systems *

Etienne Borde
TelecomParisTech
etienne.borde@telecom-paristech.fr

Peter H. Feiler Software Engineering Institute
phf@sei.cmu.edu

Grégory Haïk
Thales
gregory.haik@fr.thalesgroup.com

Laurent Pautet
TelecomParisTech
laurent.pautet@telecom-paristech.fr

ABSTRACT

Implementing adaptative systems requires to make the trade off of (i) the adaptation promptness, (ii) the amount of interferences due to adaptation, and (iii) the guarantee of data flows consistency. Furthermore, implementing critical systems requires to respect design constraints that enable to improve its determinism.

In this paper, we propose a design methodology that helps to make the adaptation trade off while respecting the critical systems design specificities. We then illustrate the usage of this methodology on an industrial use-case, using a component-based framework that generates the adaptation code. Lastly, we present and discuss the associated experimental results, and show the relative benefits of using the one or the other mode switch protocol.

Keywords

real-time embedded systems, adaptative systems, critical systems, architecture description languages, component framework, code generation, COAL, AADL, Lightweight CCM, MyCCM-HI, Ocarina

1. INTRODUCTION

Industrial real-time and embedded systems become more and more complex, offering always more functionalities while answering to very heterogeneous requirements (embeddability, fault-tolerance, timing constraints, determinism, etc...). In order to increase the capabilities of a system while answering those requirements, the notion of adaptative system has been considered as a solution to select subsets of functionalities that are active or not depending on the operational environment of the system. Design and maintenance of such systems are complex, time consuming and error prone processes. Furthermore, "critical systems" have to respect predefined safety properties all along their utilization. Implementing such systems requires guaranteeing at design time that they will operate with respect to these properties. To tackle this problem, interesting solutions based on code and formal models generation, have been proposed [12, 9]. These solutions use a subset of AADL [2] as a specification language. If they help the design of critical embedded systems, these approaches do not address specifically the implementation of adaptative systems. In order to design critical and adaptative systems, it is necessary to specify the adaptative behavior of the system while guaranteeing this behavior be analyzable at design time. AADL enables to model critical and adaptative systems thanks to the notion of **operational mode**. Besides, AADL has been used for generating the formal models of adaptative systems [10, 3]. Still, as far as we know, the problem of implementing the corresponding behavior has not been tackled. In this context, research works relative to the implementation of adaptative systems either focused on the synthesis of mode switch controllers in the scope of synchronous languages [6], or on the adaptation of non-critical systems [1]. Unfortunately, most of today's industrial complex systems does not fall under the hypothesis of synchronous languages. Thus, none of these approaches answer to our problem: how to automate the design of complex, critical and adaptative systems ?

In this paper, we propose a new design approach that relies on code generation techniques in order to implement complex, adaptative, and critical systems. This approach consists in representing the dynamic behavior of the system in a Component-Oriented Architecture Language (COAL) by (i) enumerating the system's operational modes, by (ii) representing mode switches into communicating mode automata, and by (iii) specifying which of the architecture characteristics are valid or not in a given mode. This information is then interpreted in order to produce the code corresponding to this adaptation specification. Indeed, we consider a mode as the abstract definition of a set of functionalities provided by a system or a subsystem. When adapting to new operational conditions, a system may have (i) to switch from a source mode to a target mode, and (ii) to modify the software application configuration (*e.g.* by disabling or enabling

*This research work has been lead in the scope of ANR/FlexeWare project `http://flex-eware.org`

communication links between components). This is what we call dynamic reconfiguration in the scope of this paper: adaptation mechanisms are actually handled by modifying the software applications configuration. In the remainder of this paper, we first present a basic mode switch algorithm (section 2). We then refine this algorithm in section 3 in order to answer to a specific requirement: some mode switches must be executed as fast as possible, in case of emergency for instance. In section 4, we present another refinement possibility offered by our methodology in order to guarantee the availability of consistent sets of data during a mode switch. We finally present some experimentations and associated results (section 5).

2. A MODE SWITCH ALGORITHM

When generating code to implement a reconfiguration protocol, the main issue consists in determining the reconfiguration safe state [5], which represents the moment when a mode switch can occur. In our approach, the safe state space for a mode switch is defined as the range of time during which none of the tasks impacted by the mode switch are scheduled. This corresponds to the "synchronous" mode switch protocol defined in [8]. We limit the scope of our study to "synchronous" protocols (based on rate monotonic scheduling policy) insofar as the schedulability analysis of adaptative systems is made easier using this hypothesis [8]. Thus, asynchronous protocols such as the one defined in [11] are out of the scope of our study, even if it proposes an interesting approach based on the available processor capacity.

In our approach, a task is considered as being possibly impacted by a mode switch if its execution flow depends on the value of the current mode. Thus, in order to respect hypothesis of the synchronous mode switch protocols, a mode switch must really occur when all the tasks being possibly impacted by this mode switch have completed their last execution, and none of those threads can restart until this reconfiguration is finished.

In the mode switch algorithms we propose in this paper, the current mode of a process is represented by a data shared by readers (threads impacted by a mode switch) and writers (threads configuring a mode switch). This shared data is thus protected by a read/write lock (RWLock), and must not be written while a reader thread is running. In other words, the read locking time is the full length of the impacted tasks execution. This definition ensures that a scheduled thread remains in the mode it began in until the end of its last execution, thus guaranteeing a very basic consistency rule: at a given instant, and all along their execution, all the activities of the system have the same level of information as far as the current mode of the process in which they are running is concerned. This mode switch algorithm also guarantees isolation: none of the impacted tasks has access to the mode value during a mode switch.

This mode switch algorithm constitute an implementation proposition of the "synchronous idle time protocol" described in [8]. This article also gives an equation that enables to calculate the worst case execution time associated to this mode switch algorithm.

3. REDUCING THE ADAPTATION TIME

3.1 Motivation

Depending on the mode switch emergency with regards to the priorities of the other functionalities, an adaptation process may require minimizing the adaptation delay. In this section, we propose to refine the basic mode switch algorithm in order to reduce reconfiguration time (*i.e.* the time that separates the reception of a mode switch request from the finalization of this mode transition). To reach this objective, the reconfiguration policy we present in this section enables the impacted threads to complete only their current execution, without beginning new iterations in the source mode. This reconfiguration policy is based on a "Read Write Lock Priority Ceiling Protocol".

3.2 Principles and Execution Semantics

Using the Priority Ceiling Reconfiguration Protocol (PCRP) consists in associating a ceiling priority to the mode automaton that carries out the mode switch. The main modeling constraints when using this policy in order to reduce the reconfiguration time consist in:

- giving a priority to the threads configuring the automaton instance ports that is higher than the highest priority of the threads impacted by the mode switch;
- giving to the ceiling priority a value that is higher than the highest priority of the threads configuring the mode automaton.

The execution semantics corresponding to this policy is an extension of the previous one. The main difference consists of using a "PCP read/write lock", which means that when the writer task is blocked by a set of reader tasks, these readers change of priority and get the ceiling priority defined by the software architect. Thus, readers complete their execution faster, reducing the overall reconfiguration time.

This mode switch algorithm constitute an implementation proposition of the "synchronous Minimum single offset protocol" described in [8].

4. GUARANTEEING THE DATA CONSISTENCY

4.1 Motivation

Another important requirement relative to a reconfiguration protocol deals with consistency of shared data that are accessed or written with different frequencies. For instance, let us assume that a periodic thread *T1* produces data every 10 milliseconds and that another thread *T2* reads every 50 milliseconds the 5 last sets of data produced by *T1*. We also assume that *T1* may produce data with an algorithm in mode *m1* and with another algorithm in mode *m2*. If a mode switch from *m1* to *m2* occurs after *T1* has produced 3 sets of data, *T2* would get 3 sets produced with an algorithm and 2 sets produced with another algorithm. Thus, *T2* may treat inconsistent data. We propose in this section a mode switch protocol that tackles this issue.

4.2 Principles and Execution Semantics

In our approach, using a protocol satisfying the data consistency objective consists in declaring periodic threads that have to be synchronized during a mode transition as part of the same group of synchronized threads. We call this reconfiguration protocol the "Thread Group Reconfiguration Protocol" (TGRP).

The execution semantics corresponding to this policy consists of executing the mode switch at the hyperperiod of the periodic tasks that belongs to the synchronized task group. If an impacted task belongs to different thread groups, then the considered hyperperiod period for reconfiguration equals to the hyperperiod of the union of those task groups: the task groups are actually merged. Moreover, we must insist on this point: tasks that are impacted by a mode switch and that are not part of the synchronization group are parameterized thanks to the one or the other of the previous reconfiguration policies (presented in section 2 and 3).

5. EXPERIMENTATIONS AND ASSOCIATED RESULTS

The mode switch protocols we have presented so far enable to implement adaptative systems. However, this does not provide solutions to automate the production of such systems. This is the reason why we realized a component-based framework that helps the design of adaptative and critical systems by generating the code corresponding to the adaptation mechanisms.

5.1 MyCCM-HI, our code generator prototype

MyCCM-High Integrity[1] is an experimental open source component-based framework dedicated to critical and adaptative systems [4]. From the COAL specification, MyCCM-HI generates :

- the code of the components wrappers that carries out components synchronous communications;
- the AADL model of the software architecture (including processors, buses, process, and threads);
- the synchronization code of shared data access into the components implementation code;
- the code dedicated to mode switch management.

Figure 1 sums up the generation chain of MyCCM-HI. As one can remark on this figure, the AADL model produced is used by Ocarina so as to generate the code that manages remote communications, and activations of fonctionnal chains. Piloting these generation steps and the final compilation of the executable applications, MyCCM-HI behaves like a complete COAL compiler.

Using MyCCM-HI, the software adaptation is realized by the generated code when receiving mode change requests either coming from application code or from mode automata.

[1]MyCCM-HI is available under (L)GPL at `http://myccm-hi.sourceforge.net`

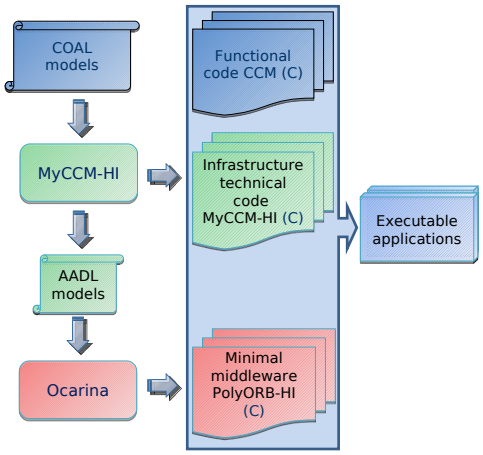

Figure 1: MyCCM-HI Generation Chain

This is quite different from usual practice that consists in detecting and performing mode switches in the functional code. The main advantage of this techniques is that it enables to reify the adaptative behavior so as to enable its analysis, and so as to improve correspondence between what is analyzed and what is implemented since both can be generated (formal model generation has been achieved using AADL [10, 3] as a specification language). Besides, code generation techniques can enforce the respect of critical systems implementation specificities [12]. In order to illustrate the benefits of using mode switch protocols we have describe in sections 2, 3, and 4, we present in this section a use-case we have implemented thanks to MyCCM-HI, and discuss the corresponding results in terms of reconfiguration time depending on the reconfiguration policy and the simulated CPU load.

5.2 Use-Case Presentation

The case-study we present in this section is a piloting system, that can operate in two operational modes: an automatic mode (A) and a manual mode (M). This system is compound of two subsystems. A localization subsystem, that provides the current position of the system every ten milliseconds, and a navigation subsystem whose behavior is different according to the current mode of the system: in automatic mode, the navigation subsystem computes the guidance commands when receiving the current position of the system from the localization subsystem; in manual mode, the navigation computes the guidance commands from orders of an end-user of the vehicle. Figure 2 illustrates the global architecture of this example. Systems boundaries are drawn with dashed boxes. The mode automaton of the piloting system is represented on the top left corner. It actually defines four modes: A represents the automatic mode, M the manual mode, while $MtoA$ and $AtoM$ are transitional modes. The behavior of the system is the following: from manual mode, the user may request to switch to automatic mode ($M \rightarrow MtoA$). If the localization subsystem is off, then the user request is rejected and the system returns in mode M. Otherwise the transition $MtoA \rightarrow A$ is fired and a communication occurs for setting the navigation subsystem to

mode A. Mode switch from A to M is similar, besides the fact that it may occur upon a failure of the localization subsystem. On this figure, connections between components are either valid in one mode, or in every mode. Dashed (respectively dotted) connections are only valid in manual (*resp.* automatic) mode. Finally, triangles represent activities with

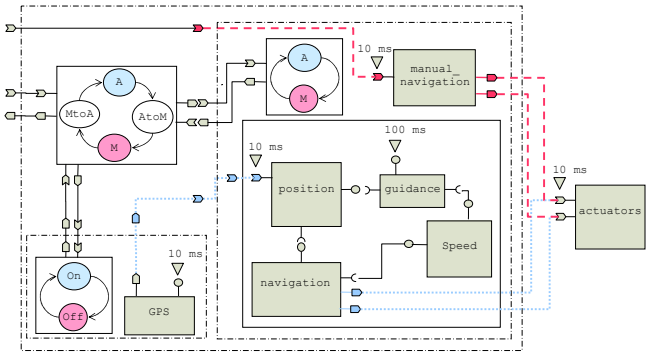

Figure 2: Pilot System and Software Architecture

their period. Indeed, in MyCCM-HI, periodic activities are used to specify the periodic activation of a given facet, while sporadic activities serve to parameterize the threads handling treatment consecutive to the reception of an incoming event (on the corresponding event sink). During a mode switch, all the threads remain actives. If sporadic threads are no longer connected to an event source, they are not dispatched. Besides, facet/receptacle connections only represent possible local operation calls. Thus, the guidance activity (periodic 100 ms) calls position component to get the last ten positions of the system, but it does not provoke an invocation to the navigation component. The navigation component is called by the position activity (sporadic 10 ms), and sends the commands to the actuators. The decision of calling or not a component through a receptacle is embedded in the component implementation code. This figure actually synthesizes the architecture description a software architect may specify in COAL so as to use MyCCM-HI code generator.

5.3 Results

Reconfiguration time. In order to measure the performance of the reconfiguration protocols in the scope of our use-case, we have defined the architecture of the complete piloting system, its mode automata, and the impact of their mode switches on the software architecture. In this experiment, we simulate the reconfiguration orders (with a rate of 210 milliseconds) and measure the time separating the request of the mode switch and the acknowledgement of the mode switch. Figure 3 provides the results of the average reconfiguration time obtained after 1000 reconfigurations (500 A $\rightarrow$ M and 500 M $\rightarrow$ A) on two different platforms. In this test, we defined four scenario:

1. we use the "read/write lock" (RWLock) reconfiguration protocol giving to the thread configuring the mode automaton instances a priority inferior to the lowest priority of the impacted threads. This configuration enables to reduce the interferences of the reconfiguration over the remainder of the application.
2. we use the priority ceiling reconfiguration protocol, giving to the thread configuring the mode automaton instance a priority superior to the highest priority of the impacted threads. This configuration aims at reducing the reconfiguration time.
3. we use the thread group reconfiguration protocol. Not synchronized but impacted thread are then parameterized with the basic mode switch protocol.
4. we use the thread group protocol and parameterize not synchronized but impacted threads with a "priority ceiling reconfiguration protocol".

We give in figure 3 the average reconfiguration time depending on the simulated CPU occupation time. Indeed, we implemented the functionalities of an autopilot but we also simulated computation time in order to increase the CPU load. We represent on the abscissa the percentage of simulated CPU occupation time over a hyperperiod of the system. This time must be added to the functional computation time, the operating system occupation time, and the framework runtime occupation time. These results have been obtained on a PowerPC (MPC5200) CPU with a frequency of 384 MHz and 256 MB of memory; the operating system being Elinos (a commercial POSIX real-time operating system).

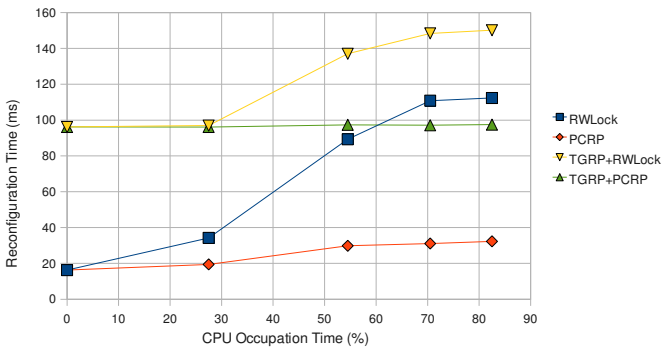

Figure 3: Reconfiguration Time and CPU Usage on Target 2

Discussion. Figure 3 shows that the experimental results correspond to the expectations we had presented in section 3 and 4:

- the reconfiguration time is reduced when using a policy that favours the reconfiguration with regards to the application. Indeed, the average reconfiguration time that corresponds to this strategy (named PCP on the figure) is always lower than the average reconfiguration time that corresponds to basic protocol (referenced RWLock on the figure).
- When we compare RWLock with PCP as the CPU load increases we notice an increasing gap between the

two curves. The reason is that in PCP the tasks synchronizing on the lock execute only once, while in the case of RWLock under higher loads as the completion time of the longer period task increases a shorter period task may dispatch again and be able to acquire the lock as reader.

- The gap between the protocols without TGRP and the protocols combined with TGRP reflects the synchronization group hyperperiod: it is important to note that the reconfiguration time, when using the thread group reconfiguration protocol depends on the hyperperiod of the group of threads that have to be synchronized.

Moreover, this figure illustrates the relative cost, in terms of reconfiguration time (*w.r.t* the CPU load), of using the one or the other of the presented strategies. In our use-case, we finally chose the PCP reconfiguration protocol: 100 milliseconds for the mode switch was not an acceptable performance, and data set during a mode switch from manual to automatic (and vice versa) does not require to keep in storage any data flow.

Compactedness of the generated code. In addition to these results, we add that the obtained application needs less than 900 KB to execute (it contains 11 tasks, 9 components and 3 mode automata). This result constitutes an important breakthrough with regards to memory footprint obtained with CORBA [7] based component frameworks. Moreover, this use-case ilustrates the gain for development teams: from 836 lines of COAL specifications, 20 248 lines of C have been generated. These results also illustrate the compactedness of the generated code.

6. CONCLUSION AND FUTURE WORKS

In this paper, we have presented a framework that enables (i) to model mode-based reconfiguration, (ii) to parameterize the behavior of the system under reconfiguration, and (iii) to generate code that implements the mode switch mechanisms and their impact on the software architecture (so called software dynamic reconfiguration). We also presented a set of reconfiguration policies that enables to perform the necessary tradeoffs of a reconfiguration protocol. For all those reconfiguration strategies, we have provided experimental results that confirm our expectations in terms of reconfiguration time. Besides, these results have been obtained on an industrial use-case (described in this paper) that illustrates the needs for reconfiguration, as well as the needs for different reconfiguration strategies. Furthermore, optimizations are still possible. For instance, an important optimization will consist of parameterizing not only the reconfiguration policy associated to whole automaton instance, but to be able to parametrize each of its modes transitions independently. Besides these optimization possibilities, the work we have presented in this paper opens a broad spectrum of perspectives in the domain of real-time and embedded systems modeling. For instance, this could be considered as an entry point for the formal verification of reconfigurable applications, or as an extension to the domain of fault tolerance modeling. Last but not least, this work may serve as a basis to improve and/or create standards dedicated to the specification of reconfigurable yet critical systems, and will lead to propositions to improve the SAE/AADL and OMG/CCM standards.

7. REFERENCES

[1] *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), 18-20 May 2005, Seattle, WA, USA*. IEEE Computer Society, 2005.

[2] S. A. E. AS5506. Architecture analysis and design language (aadl). Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, November 2004.

[3] D. Bertrand, A.-M. Déplanche, S. Faucou, and O. H. Roux. A study of the aadl mode change protocol. In *ICECCS '08: Proceedings of the 13th IEEE International Conference on on Engineering of Complex Computer Systems*, pages 288–293, Washington, DC, USA, 2008. IEEE Computer Society.

[4] E. Borde, G. Haïk, and L. Pautet. Mode-based reconfiguration of critical software component architectures. In *11th ACM - IEEE International Conference on Design Automation and Test in Europe*. DATE'09, April 2009.

[5] K. M. Goudarzi and J. Kramer. Maintaining node consistency in the face of dynamic change. In *In Proceedings of the Third International Conference on Configurable Distributed Systems*, pages 62–69. IEEE Computer Society Press, 1996.

[6] O. Labbani, J.-L. Dekeyser, and P. Boulet. Mode-Automata based Methodology for Scade. In *Hybrid Systems: Computation and Control (HSCC05)*, Zurich, Switzerland, 03 2005.

[7] O. M. G. (OMG). Common object request broker architecture : Core specification, version 3.0.3. Technical report, OMG (Object Management Group), March 2004.

[8] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197, 2004.

[9] X. Renault, F. Kordon, and J. Hugues. From AADL architectural models to Petri Nets : Checking model viability. In *12th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'09)*, pages 313–320, Tokyo, Japon, mar 2009.

[10] J.-F. Rolland, J.-P. Bodeveix, M. Filali, D. Chemouil, and D. Thomas. Modes in asynchronous systems. In *ICECCS '08: Proceedings of the 13th IEEE International Conference on on Engineering of Complex Computer Systems*, pages 282–287, Washington, DC, USA, 2008. IEEE Computer Society.

[11] L. Sha, L. Sha, R. Rajkumar, R. Rajkumar, J. Lehoczky, J. Lehoczky, K. Ramamritham, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1:243–264, 1988.

[12] B. Zalila, L. Pautet, and J. Hugues. Towards Automatic Middleware Generation. In *11th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'08)*, pages 221–228, Orlando, Florida, USA, may 2008.