Timing Fault Detection for Safety-Critical Real-Time Embedded Systems

Sébastien Faucou, Anne-Marie Dplanche and Yvon Trinquet
Institut de Recherche en Communications et Cybernétique de Nantes
1, rue de la Noë, BP92101 44321 Nantes Cedex 3, France
{faucou|deplanche|trinquet}@irccyn.ec-nantes.fr

Abstract

On the one hand, a major aspect of dependability for real-time embedded systems is the respect of timing requirements. On the other hand, the complexity of modern real-time embedded system implies the need for new design process focusing on high-level features, such as architecture-based design. In this paper, we show how to integrate a timing fault detection technique in such a design process. Our approach is based upon the CLARA ADL (Architecture Description Language). This language allows to describe applications which can be easily implemented thanks to a distributed middleware designed on top of the OSEK/VDX real-time kernel.

1 Introduction

The reliability and the low-cost of today electronic components have led many industries to increase the part of embedded systems in their field. A representative example is the automotive industry which integrates more and more in-vehicle embedded "Electronic Control Unit" to increase safety and comfort by providing new features. For these embedded real-time control systems, one of the major dependability requirements is safety (the system must not damage its environment). In this paper, we focus on a special kind of safety-related faults: timing faults. For a real-time system, its service is correct not only if the results it delivers have good values but if the dates where they are produced are also good. Actually, in such a context, the violation of a timing constraint can be safety critical to the environment.

Because these systems become more and more complex, they now exhibit more "classical" requirements: flexibility, reuse, interoperability, etc. One way to handle these requirements is to adopt an architecture-based design process, which makes it possible to reason about the architecture level of design [10]. Our goal is to investigate the relations between the architectural design of a real-time application and the verification of its timing requirements, and we show how a timing analysis can be conducted at this level. Such an approach, while not requiring a detailed algorithmic knowledge of the application enables to detect and thus to correct design mistakes early in the life cycle. This timing fault detection step is based on a behavioral analysis. Obviously, it does not preclude other means for dependability and particulary fault-tolerance mechanisms.

To perform a timing analysis, the whole system has to be taken into account: application software, execution platform and environment. Our work being based on the CLARA ADL, we explore in a first time the implementation of CLARA descriptions. To facilitate this process, we have developed a dedicated distributed middleware on top of the OSEK/VDX real-time platform. In a second time, we build a fine grain model of this implementation, which takes into account details of the low level software. The simulation of this model allows to observe the timing behavior of the candidate architecture and to validate it w.r.t. the specified timing constraints, that is to detect timing fault occurrences and locate their sources so as to design a new candidate.

2 Context

2.1 In-vehicle embedded systems

Today, vehicles include more and more electronic systems and features. Some of these new functionalities run across different sub-systems which have to communicate. Moreover, this kind of systems is developed for a wide range of products. As a consequence the software and hardware requirements have evolved and include now flexibility, portability, reuse, hardware/software independence, etc.

For the software aspects, one can cite as an illustration the OSEK/VDX architecture [9]. It is a joint project of European car industries the aim of which is to propose a standard platform for in-vehicle embedded applications. It is made up of different specifications: OSEK OS (scalable kernel for real-time embedded systems), OSEK

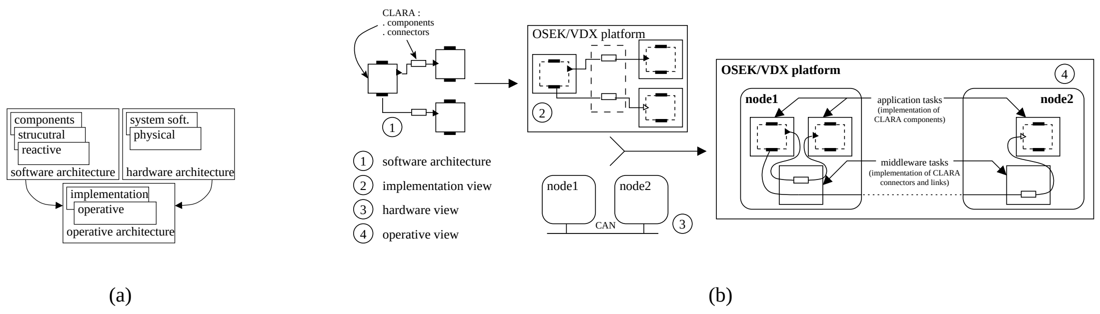


Figure 1. Design process of an operative architecture with CLARA.

COM (application-level communication protocol), OSEK NM (network management), etc.

For the hardware aspects, one of the major requirements is to provide a flexible way to interconnect subsystems. To address this requirement, network protocols have been developed in order to build distributed systems around multiplexed bus that offer an acceptable QoS. These protocols include Controller Area Network (CAN, based on CSMA/CR), Time Triggered Protocol (TTP, based on TDMA), Local Interconnect Network (LIN, master/slaves), etc. To experiment our approach, we will consider systems composed of OSEK/VDX nodes interconnected through a CAN bus.

2.2 Architecture description of real-time applications

Software architecture is a field of interest for scientist since the mid 90's [10]. It is the domain of software engineering dedicated to high-level design. Part of the work on software architecture has been carried out on ADL, which aims at describing a software architecture as the interconnection of *components* (processing entities) and *connectors* (communication entities). Being formally defined, an ADL provides with the possibility to perform architecture-level analysis: liveness [2], reliability [12], etc.

The CLARA ADL, which is dedicated to real-time applications, has been defined in our team [6]. CLARA components are active objects (they own an execution flow) and the set of predefined connectors covers synchronous and asynchronous event signalling and message passing mechanisms. The semantics of CLARA is defined with (time) Petri nets and dedicated static verification techniques allow to perform formal analysis of architecture descriptions. Exhaustive timing analysis techniques are also used, but they do not allow to take into account most of the implementation aspects. As the timing behavior of a system relies heavily on its implementation, this is a severe restriction. We propose in this paper a simulation-based technique for the timing analysis of CLARA architecture description which takes into account the effective implementation of the system, including hardware and system software details.

CLARA offers different views of software, hardware and operative architectures (resp. SA, HA and OA), each one being relevant for a specific aspect: component (component specification), structural (component interconnection) and reactive views (interface of the environment and components activation mechanisms) for the SA; physical view (description of the physical architecture) and system software description for the HA; implementation and operative views for the OA. The implementation view is the mapping of the SA onto the execution platform native objects and services, whereas the operative view is the mapping of the implementation view onto the physical view. A summary of this organisation is given fig. 1.

To allow a straightforward design of the implementation view, we have defined a set of mapping rules from CLARA high-level artefacts onto OSEK/VDX native objects (e.g. components are mapped onto OSEK OS tasks). Together with these rules, we have designed a middleware which offers high-level services to the application layer for the mapping of connectors and links. To preserve the benefits of the software architecture approach w.r.t. software evolution and reuse, only four services can be used: `Send` and `Receive` for data transmission and `Signal` and `Wait` for event signalling. The knowledge of the architecture allows the middleware to use the mechanisms and data structures corresponding to the specified connector instance. As embedded real-time systems are static (i.e. all objects are known before runtime), the middleware uses static routing tables to perform its services.

For each node in the system, a task is in charge of handling intra-node interactions, delivering incoming messages and sending outcoming messages. It communicates with application tasks using OSEK OS event notification mechanisms and OSEK COM messages (middleware services are macros based on OS services and behave as transactions). The connectors of the CLARA description are “inlined” in the middleware. This organization is illustrated in fig. 1(b). A prototype has been developed in C.

3 Architecture-level timing analysis

3.1 Related works

Architecture level timing analysis does not come as a replacement for lower-level timing analysis that can be performed once the detailed design step is achieved. It aims at validating early high level design: application partitioning and structure, activation mechanisms, allocation and scheduling of tasks and frames, etc. To perform such an analysis, a narrow approximation of the internal behavior of the components must be known (including estimation of computing times, which can be obtained from a WCET analysis for pre-existing components and by a priori evaluation for other ones).

There is not a lot of architecture-based tool-set dedicated to the design of real-time applications. One can cite the BASEMENT framework [7] for in-vehicle embedded systems and the MetaH ADL [3] for avionics embedded systems. For the timing analysis, BASEMENT comes with an off-line scheduler and a discrete event system level simulator tool. The simulator uses an emulator of the BASEMENT platform which executes the effective code of the components on a modified BASEMENT kernel. Hence, it can only be used after the completion of the detailed design step and is not adapted to early validation of the high level design. To handle timing requirements at high level, the MetaH tool-set includes a schedulability analyzer, which works on analytical models of the tasks and messages sets. These models are refined along the design process but are constrained by the analytical method used. In order to handle more complex task model and to formally verify critical application components (e.g. scheduler, bus driver, etc.) the MetaH team has also been working on linear hybrid automata models. This technology is unfortunately not mature enough to be applied to a whole architecture.

Our goal is to validate the architecture of a system w.r.t. its timing requirements (e.g. basic and end-to-end deadlines, cadence, etc.). It is common that a time-constrained functionality is performed through the cooperation of several tasks, eventually distributed over different nodes. Thus, the middleware, OS services and eventually the networks are involved in its achievement. As the use of OS or middleware services is of high influence with the application (time overhead, rescheduling, update of the status of system objects, etc.), it must be taken into account when analyzing the (timing) behavior of the system. In order to fulfill this goal we adopt a method based on the simulation of a model of the system built from detailed models of COTS components (e.g. the execution platform) and high level model of application components which are not fully defined. This approach is similar to [5] but can handle a broader range of task behaviors, in order to cover the possibilities of CLARA.

3.2 System modeling

For the modeling and simulation step, we have selected the *Object*GEODE framework from Telelogic [1], which is based upon the SDL formal description technique. SDL uses jointly Asynchronous Communicating Finite State Machines and Abstract Data Types paradigms for system modeling, so as to handle concurrency as well as data modeling. It is known to being unadapted to real-time system specification [4] but we use it as a system description language and we are not penalized by its limitations. Moreover, the *Object*GEODE simulator tool [11] offers some extensions, especially a semantics which makes it possible to control time progression.

In order to automate the translation of the OA to an analyzable model, we have developed a set of predefined SDL process types: CPU, CAN stack and network, OSEK OS, OSEK COM, interupt service routine, middleware task, etc. These models include both functional and timing characteristics. To build the system model, they are instantiated and connected following predefined rules. The remaining work resides in the injection of components behavior and low level software configuration in this skeleton. Finally, the system model must be closed by a model of the environment so as to facilitate the simulation.

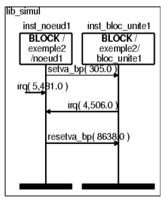
3.3 Analysis technique and results

For the verification step, *ObjectGEODE* comes with a model-checker. Unfortunately, it does not allow to verify quantitative timing properties (although ongoing works address this [8]). Moreover, model-checking techniques are confronted with the state-space explosion problem when the number of variables in the model is huge. As a consequence, we use presently the tool in simulation mode.

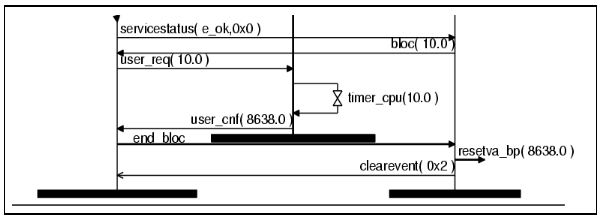
CLARA allows to express the timing constraints of the system on the architecture views. These constraints are translated into observer automata [1] which are merged to the SDL model. When an observed event occurs during the simulation run, the concerned observers change their states and it is thus possible to detect wrong sequences.

As an output, the simulator generates the trace of signals exchanged during the run. From this trace, Message Sequence Charts (MSC) can be generated at different levels of details by selecting the SDL entities of interest. At the highest level, it is possible to isolate only the signals exchanged

[1]Telelogic web site: `http://www.telelogic.com`.



(a)



(b)

Figure 2. Two MSC generated from the same run at different detail levels.

between a node (seen as a black box) and the environment for a particular functionality. At the lowest level, it is possible to include OS-level signals, especially scheduling signals resulting from middleware and OS service execution: preemption, release, etc.

Figure 2 shows two MSC obtained after a simulation of a factory control system case study. The first (2(a)) is the highest level one: the two entities are node 1 of the control system and the environment. Signals are filtered so as to see only the ones implied in the constrained functionality under study (here the control of valve `va_bp`). This high level MSC underlines a violation of a timing requirement (deadline violation between irq 4 and `resetva_bp`), detected during the run by the observer associated to this constraint. The second (2(b)) is (a small part of) the lowest level one, the study of which allows to find the source of the timing fault (a priority allocation mistake, allowing a time consuming sampling task to be active for a too long time). To correct this mistake, there are many options the consequences of which have to be analysed by running new simulations: changing the priority assignement, changing the allocation of tasks to node, changing activation mechanisms so as to suppress the conflict, using a simple interrupt service routine to handle the valve, etc.

4 Conclusion

We have exposed an architecture-level timing analysis technique which is usefull to perform early validation of high level design w.r.t. timing constraints. It is based on simulation at system level. For the results to be accurate, the model must be close to the effective system and we adress this issue (i) by offering a middleware and a set of rule for the implementation of architectural artefacts and (ii) by taking into account in the models the knowledge of the runtime platform (hardware architecture, OS, network, middleware, etc.). Aiming at being applied to high level design, it is flexible enough to cover a wide range of real-time application style.

In order to provide with a comprehensive architecture design process, a complementary work should investigate the mapping of event traces collected during the simulation onto the architecture elements. This work should also be directed towards the identification of “good” and “bad” design practices (w.r.t. timing requirements) in order to assist the architect in his corrective work.

References

[1] B. Algayres, Y. Lejeune, and F. Hugonnet. GOAL : Observing SDL behaviors with GEODE. In *Proc. of SDL Forum 95*. Elsevier, 1995.

[2] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.

[3] P. Binns and S. Vestal. Formalizing software architectures for embedded systems. In *Proc. of EMSOFT 2001*. Springer, 2001.

[4] M. Bozga, S. Graf, L. Mounier, I. Ober, and D. Vincent. SDL for real-time: what is missing. In *Proc. of SAM’2000*, 2000.

[5] P. Castelpietra, Y. Song, F. Simonot, and O. Cayrol. Performance evaluation of multiple networked in-vehicle embedded architecture. In *Proc. of WFCS’2000*. IEEE IES, 2000.

[6] E. Durand and A. Déplanche. CLARA - An Architecture Description Language for Real-Time Applications. Technical report, IRCCyN, Nantes, March 1999. (in french).

[7] H. Hansson, H. Lawson, M. Strömberg, and S. Larsson. BASEMENT: a Distributed Real-Time Architecture for Vehicle Applications. *Real-Time Systems*, 11(3), 1996.

[8] I. Ober and A. Kerbrat. Verification of Quantitative Temporal Properties of SDL Specifications. In *Proc. of SDL Forum 2001*. Springer, 2001.

[9] OSEK Group. OSEK/VDX OS 2.2, COM 2.2.2, OIL 2.3, NM 2.5.1. `http://www.osek-vdx.org`, 2001.

[10] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[11] Verilog. *ObjectGEODE SDL Simulator - Version 4.0*, 1999.

[12] A. Zarras and V. Issarny. Assessing Software Reliability at the Architectural Level. In *Proc. of ISAW 2000*, 2000.