Abstract

This document presents a simulation-based method for the development of real time embedded systems. This kind of system is really sensitive to the communication with external environment and the Quality of Service (QoS) provided by the used target. So, the method is MDA-based and proposes to separate the concerns, introducing a communication layer with the outside environment, an application layer and a connection layer between these two layers. Then, it is possible to emulate each part on the system in order to validate it.

1 Introduction

Real Time Embedded Systems (RTES) become more and more complex. This complexity creates a need for development methods that decrease development and simulation time.

A RTES builds, in real time, a view of a controlled process (Inputs) by using sensors (cf. figure 1). Then it computes the necessary commands (Outputs) in order to control the environment. Finally, commands are performed using actuators. Since the environment has its own behaviors, all operations must be performed respecting Quality of Service (QoS) constraints such as a minimum frequency or a maximum delay. In consequence, the communication protocol with the environment is a key point of such systems and has to be precisely specified; in a functional way as well as in a non-functional way. So, due to their specific and contextual constraints (imposed by the interaction with the environment) RTES are dedicated systems developed in an ad-hoc and hand-made manner. Of course, this type of development leads to difficulties regarding to early application validation usually based on a emulated environment and a virtual target.

Figure 1: Representation of a RTES

Software engineering tends to deal with these problems. Particularly, the introduction of Object Oriented (OO) methods and more recently of Model Driven Development (MDD) are good helps for application evolution [12]. One of the MDD approaches, MDA (Model Driven Architecture: [13]), is particularly focused on the separation of concerns. To separate the concerns, MDA introduces three models: PIM\(^1\), PM\(^2\) and PSM\(^3\), where the PSM results from

---

\(^1\)Platform Independent Model
\(^2\)Platform Model
\(^3\)Platform Specific Model
the transformation of the PIM and the PM. MDA is a promising model-based development method for general purpose systems.

This paper presents how the specificity of RTES can be integrated in a MDA-based approach in order to allow quick development and simulation. Since it is significant to dissociate the concerns relating to the communication protocol, the proposed approach is based on three layers (target layer, application layer and adaptation layer). Each one is described by a UML2.0 model (Unified Modeling Language: [15]).

After a presentation of the context, we describe the goal of each layer and how they are linked. Then, we present the models and specifically, we detail the Adaptation Layer Model (ALM). As a conclusion, we explain the uses of the approach for simulation and we illustrate it with a case study.

2 Context

The Object Oriented (OO) paradigm is widely acknowledged by the industrial world and reinforced by the emergence of a standard (UML). The main interest of OO approaches is their ability to improve the quality of software [10, 19]. But OO approaches lack of architecture structuring capacities and semantic information. Initiated by the OMG 4, MDA is proposed for system development to increase portability, interoperability and reusability through architectural separation of concerns. MDA uses models for system representation. A model, defined by its metamodel, is a system abstraction focusing on a specific concern (in MDA, the PIM focuses on the application behaviors and the PM on the execution environment). A whole system (a PSM in MDA terminology) is obtained by models transformation (mapping between PIM and PM in MDA terminology). Following a MDA approach, this paper considers the target model (i.e. a Sensor/Actuator Model: SAM) as the platform model. The question is then how to describe the PIM (i.e. a Sensor/Actuator Independent Model: SAIM) and how to correctly perform the mapping to produce the resulting PSM (i.e. a Sensor/Actuator Specific Model: SASM). With the objective of ensuring consistency between models, a description of the QoS is necessary for each model [2]. There are two main kinds of QoS specification: the required QoS, and the provided QoS. The UML QoS profile [16] states that a system is consistent if and only if $QoS_{provided} \geq QoS_{required}$. The QoS specification helps making restrictions on deployment platforms (without platform knowledge). We classically consider three QoS characteristics for RTES: delay, arrival law and precision. Once all QoS specifications are done, an admission test is mandatory to ensure that the provided QoS satisfies the required QoS. In RTES domain, the QoS specification has to be a priori validated; so this test is done at modeling time.

3 Related Works

Frankel et al. [6] and Wampler [4] use MDA successfully for Web Services; in the same way Caceres et al. [18] use MDA for financial systems. More generally, MDA is successfully accepted for general purpose systems. In this kind of system, the target (S/A) is characterized by classical user interface devices such as keyboard, mouse and screen. It is almost the same and during design it is implicit or virtualized (virtual mouse, virtual screen). So, S/A simulation and adaptation is not a key point of such applications. There are few MDA-based implementations for dedicated systems where correctness strongly relies on hardware performances such as RTES [1, 2]. Boulet et al. [17] propose the use of MDA for System-on-chip design. This MDA adaptation of the "Y-approach" introduces an association-model in order to model mapping between software and hardware descriptions. But this approach considers machines and buses as the PM and not S/A dependencies. Houberdon et al. [8] introduce the S/A in the use of MDA for systems dedicated to process control, but do not provide elements on how to describe and interpret QoS requirements, which is a crucial point for simulation and validation of RTES. Bordhar et al. [3] propose the use of UML and OCL [14] constraints in order to specify QoS and to realize the admission test. But this last approach is neither MDA-adapted nor
4 The MDA-based approach

The first subsection describes the objectives of the proposed approach more precisely while the second one presents the architecture and tool boxes proposed for model descriptions.

4.1 Architecture principles

In order to allow simulation and deployment of a same software without any changes, the proposed software engineering paradigm deals with separation of concerns between target, application and communication protocol. On the one hand, the description of the behaviors is based on a desired view of the environment (I/O: data, event, command). It models the state of the environment through data, events and requirements on the environment changes through commands. On the other hand, the S/A drivers represent a specific target providing a specific view of the environment (driver I/O). To perform mapping of the desired view of the environment (I/O) to the specific view given by a target (driver I/O), we introduce the ALM which may be viewed as a connection model between the two layers. It realizes the "smart" link (functional and extra-functional) between I/O and driver I/O. This adaptation layer is independent from the other concerns and encapsulated in a specific model.

To achieve the software engineering paradigm, five questions have to be answered:

1. What Input and Output (I/O) information is useful for the behaviors ?
2. What are the I/O QoS constraints required to ensure the validity of the application ?
3. What is the QoS provided by driver I/O of a specific target ?
4. How can information needed by the application (I/O) be produced from driver I/O of a specific target ?
5. Is it possible and if so, how driver I/O of a specific target can provide correct QoS ?

The answers to these five questions are given through three models representing the three layers of the system (cf figure 2). The SAIM (Sensors/Actuators Independent Model) answers to questions one and two. Its metamodel is composed of elements characterizing families of Input/Output (I/O). Moreover, it possesses a specific profile that allows specifying QoS constraints required for the correctness of the system. Once this is done, a SAM (Sensors/Actuators Model) has to be chosen and characterized, from QoS point of view, for the deployment of the system (Question 3). Then, ALM elements drive us to answer to questions four and five. To do that, the adaptation layer is composed of three elements (DAE, EAE, CAE). Each one is dedicated to a specific type of I/O.

4.2 Architecture

4.2.1 SAIM Elements

Behaviors This element encapsulates the application functional activity and is seen here as a black box.

---

5 Data Adaptation Element
6 Event Adaptation Element
7 Command Adaptation Element
Data  A data is one or more encapsulated value(s) representing a part of the environment information needed by the behaviors. It appears in the SAIM model and, so, represents high-level information without any production details (example: a speed data does not specify the type of sensor(s) used to acquire the information). We differentiate two kinds of data: the continuous data, representing values which can be measured on a scale; and the categorical data, representing data whose values are part of a finished list. They possess the following generic functional services:

- \( T\_Value \) Get data()
- \( \text{Set data}(T\_Value \ new\_data) \)

and the following QoS characteristics:

- sampling law: specifies inter-arrival time between two consecutive set data() calls
- delay: specifies the new data age when set data() service is called
- precision: specifies the absolute error between a physical data value and the new data corresponding value (only for continuous data)

Event  An event is a representation of event occurrences required by the system. It represents a high-level event without any production details. We differentiate three kind of events: counted events, boolean events and fugacious events. They all possess one generic functional service:

- New event()

Generic QoS characteristics used for events are:

- arrival law: specifies inter-arrival time between two new event() calls
- delay: specifies time between a physical stimuli and a new event() call

Command  A command represents an action which has to be performed on the environment. It also represents high-level information without any realization detail, and like data, command can be continuous or categorical and possess at least the following functional services:

- \( T\_Value \) Get cmd()
- \( \text{Set cmd}(T\_Value \ new\_cmd) \)

QoS characteristics are the following:

- refresh law: specifies inter-arrival time between two set cmd() calls. (We can notice that this is the only provided QoS characteristic in the SAIM)
- delay: specifies acceptable time between a set cmd() call and its physical realization
- precision: specifies absolute error between a new cmd value and the corresponding realized value (only for continuous command)

In this study, we specify all QoS characteristics thanks to intervals [min; max].

4.2.2 SAM Elements

driver I/O  The driver I/O elements encapsulate services and QoS provided by drivers of S/A. driver data and driver event acquire information from the environment through sensor(s) while driver cmd realizes actions on the environment through actuators. Drivers provide low-level information destined to the adaptation layer. They possess the same services than the SAIM elements (set()/get()). Moreover, QoS characteristics are the same than those presented in the previous section (data \(\rightarrow\) driver data, event \(\rightarrow\) driver event, command \(\rightarrow\) driver cmd). In addition to these statistical characteristics, we add timestamps information, useful for the adaptation layer at runtime. We can notice that only the refresh law characteristic is a required one, the others are provided ones. Even if the evaluation of the SAM QoS is not trivial, it is possible to analyze and acquire them [5]. The remainder of this paper considers that they are provided.

4.2.3 ALM elements

The ALM is the association of several DAE, EAE and CAE. Each adaptation element possesses at least two services called during execution: interpret and QoS adapt; and one other called during modeling: admission test.

Interpret service encapsulates the human knowledge specifying how low-level pieces of information are used in order to produce high-level one in input
(cf. figure 3 (a)) and how high-level information is realized by low-level ones in output (cf figure 3 (b)). To realize this service, first, a formatting is mandatory (expected for event). It allows casting driver data value or command value in order to manipulate them consistently. It consists in unity change and/or reference marks change. The service signature is:

- \( T_{\text{value}} \) \( \text{format} \) \( (T_{\text{value}}) \)

Once the formatting, if necessary, has been done, interpretation can start. Interpretation realized by data and command adaptation elements is a semantic adaptation since it gives/changes semantic of drivers information. It is simple if sensor(s) or actuator(s) correspond directly to high level information described in the SAIM (→ interpretation is empty). On the contrary, it is very complex if there is few relation between informations needed/provided by the SAM and high level information of the SAIM. So, in order to help software designer and decrease development time, the approach proposes a tool box composed by generic interpretation policies. These interpretation policies are modeled via timed automate. In this paper we focus on two classical input interpretation policies. The first one (see fig 4) is dedicated to information aggregation in order to produce a data. Once all sensors pieces of driver data are captured, they are computed to create a data. The second automate models an event oriented protocol. It waits for a specific sequence of driver event in order to produce an event (cf. figure 5).

QoS evaluation It is important to notice that interpretation impacts the QoS provided by the SAM. A new associated QoS has then to be evaluated. Because one or more new pieces of information are constructed, this evaluation is based on the following parameters: QoS provided by associated S/A from the SAM and type of adaptation policy (box type) used.

The following equations describe how to evaluate resulting QoS characteristics for data aggregation (cf. figure 4) and the event protocol creation automate (cf. figure 5). Due to lack of space, we explain only the more complex characteristic equation: the delay. For data aggregation, we consider that the resulting delay is the maximum, respectively the minimum age of all assembled data. In the worst case, used data have been produced in the previous period. In the best case, all used data are produced at the same time. This information gives the delay induced by the data assembly statechart protocol. It is then necessary to add the delay of the data and the time spent in the assembly function. In following construction enumeration, arrival law, sampling law, delay, and precision of driver \( J_i \) are respectively specified by \( [A_{me_i}; A_{Me_i}] \), \( [S_{me_i}; S_{Me_i}] \), \( [D_{me_i}; D_{Me_i}] \), and \( [P_{me_i}; P_{Me_i}] \). Then the resulting characteristics are respectively specified by \( [A_{ms}; A_{Ms}] \), \( [S_{ms}; S_{Ms}] \), \( [D_{ms}; D_{Ms}] \), and \( [P_{ms}; P_{Ms}] \). \( C_{\text{construct}} \) is the time spent for the construction of data or event. Finally \( N \) represents the number of associated driver I/O.

1. QoS construction for data assembly

- \( S_{Ms} = \max_{1 \leq i \leq N} (S_{Me_i}) \)
- \( S_{Ms} = \max_{1 \leq i \leq N} (S_{me_i}) \)
\[ DMs = \max(\max_{j=1}^{N} (SMe_j + DMe_j); \quad \sum_{k:SMe_k < SMs} DMe_k ) + C_{construct} \]

\[ Dms = \max_{i=1}^{N} (DMe_i) + C_{construct} \]

\[ Pms: \text{depends on the assembly’s function. For an average function:} \]
\[ Pms = \sum_{i=1}^{N} (pme_i) / N \]

\[ PMs: \text{In the best case, } PMs = 0 \text{ because precision is given with } \pm \text{ values} \]

For delay evaluation of event protocol creation, we consider here that the delay is the time spent between acquisition of the last received driver_event and the event generation.

### Figure 5: No timed event protocol statechart

2. QoS construction for event’s protocol

- \( AMs = \sum_{i=1}^{N} (AMe_i) \)
- \( Ams = \max_{i=1}^{N} (Ame_i) \)
- \( DMs = \max_{i=1}^{N} (DMe_i) + C_{construct} \)
- \( Dms = \max_{i=1}^{N} (Dme_i) + C_{construct} \)

After the provided QoS of the adaptation layer constructed, an admission test has to be realized.

**Admission test service** This service is used during modeling. It consists in a comparison between the provided QoS characteristics from the adaptation layer and the required QoS characteristics from the SAIM. It has to be done for each characteristic. Because we use intervals, the generic QoS\(_{\text{provided}} \geq \) QoS\(_{\text{required}} \) admission test is defined by:

\[ [\text{Min}_{\text{provided}}; \text{Max}_{\text{provided}}] \subseteq [\text{Min}_{\text{required}}; \text{Max}_{\text{required}}] \]

Once admission test has been done, there are three solutions; or there is consistency and there is no need to adapt QoS, or there is not and: 1) QoS\(_{\text{adapt}}() \) service has to be specified in order to reach consistency; 2) consistency can not be reached and it is impossible to use this SAM.

**QoS\(_{\text{adapt}} \) service** This service allows modifying QoS provided by an input adaptation element (DAE and EAE). Since it is possible to adapt only provided characteristics, in output, only refresh_law is adaptable. The main realizable QoS adaptations are:

- **delay:**
  1. realization of a fix delay (thanks to "now" and timer use)
  2. increase of the delay (thanks to timer use)

- **arrival_law/sampling_law:**
  1. reduction of the (arrival/sampling)law\(_{\text{max/min}} \) (thanks to interpolation or repetition)
  2. augmentation of the arrival/sampling\(_{\text{law}_\text{max}} \)
  3. realization of a periodical arrival/sampling_law (thanks to timer: cf figure 6)
  4. decrease of a periodical arrival/sampling\(_{\text{law}_\text{max}} \) (interpolation)
  5. increase of a periodical arrival/sampling\(_{\text{law}} \) (repetition)

- **precision:**
  1. reduction (rounding, cutting, ...)

- **refresh_law:**
  1. augmentation (repetition)
  2. realization of a periodical refresh law (thanks to timer)

It is important to notice that like the interpretation modifies resulting QoS, the QoS adaptation impacts all QoS characteristics. For example, use of interpolation to reduce maximum sampling_law impacts on the maximum delay and the precision provided for the data. Due to a lack of space, identification of the QoS provided by each adaptation box is not presented here. Figure 6 gives a view of the automate associated with realization of a periodical arrival law: when a new data is constructed, it is memorized and propagated in the SAIM only when the timer expires.
5 Simulation methodology

We propose now a methodology based on the previous models suitable for early simulation and validation of RTES. The method defines three main steps.

During the first step, the SAIM has to be validated regarding to its required QoS. The description of the SAIM I/O and their associated QoS allows realizing an environment emulator able to provide input signals corresponding to the description of each input and evolving regarding to the output created by the application. The emulator may be implemented using tools like matlab [11]. In case of a simple behavior for the environment (no relation between a new output and the next input), an User Interface can emulate the external environment. In order to model the expected QoS, it is necessary to correctly adapt the QoS of the emulator adding QoS adaptation boxes.

Once the SAIM has been validated, the communication part has to be specified regarding to the expected target. During the second step, the ALM is built using an emulated SAM corresponding to the target. In order to be closer to the QoS provided by the real target, it is also necessary to correctly adapt the QoS of the emulator. The SAM is then associated with QoS adaptation boxes. This step ensures that the communication protocol is correctly defined and so performs the SASM validation. Moreover, in order to simulate different behaviors for the target (normal, faulty), a simulation UI is introduced which interacts with the emulated SAM and its QoS adaptation boxes.

Then for the third step, linking the real target with the application needs to remove previously introduced QoS adaptation boxes of the SAM and the simulation UI. Then the emulated SAM is substituted by the real S/A drivers. No change in the application behaviors (SAIM) is needed.

We can notice that it is possible to have an incremental approach mixing steps two and three: just a part of the target is emulated. For example, when first testing an exploration robot controlled by a joystick, the real joystick driver may be used whereas the robot is emulated. The next section presents a case study of a terrestrial exploration robot, using this approach and allowing an incremental development cycle.

6 Case study

This section presents how the presented approach has been used for the Josefil challenge [9]. Josefil is a terrestrial exploration robot whose goal is temperature measurement. It has to move into a well known space area (map known a priori). It receives a position to reach, moves to this position, acquires and saves temperature and waits for a new position to reach. It also provides a video of its mission. In order to apply the proposed approach, the first step is the construction of the SAIM. The robot represents the environment through five main high level data: "actual position", "actual orientation", "position to reach", "temperature" and "current view"; and one main event: "on/off". It interacts on the environment through four main high level commands: "speed to reach", "rotation to reach", "temperature to save" and "image to transmit".

The control robot behaviors (SAIM) are formally implemented using SDL [7]. Josefil provides an emulator providing the same functionality than the real robot (cf. figure 7: http://www.wanyrobotics.com/softwareLab.html). Thus, the simulation is possible through Object-Geode simulator [20].

So the second step introduces the emulated SAM. The ALM is built to adapt the SAM services to the
previously modelled and validated SAIM. Since the Josefil emulator does not give a realistic view of the QoS provided by the real target, it has to be QoS adapted. For example, since the emulated SAM gives a perfect position (i.e. with a maximal precision), we introduce a QoS adaptation box to reduce the precision of the position. Moreover, the video information is given at high arrival rate. To be realistic, we introduce another QoS adaptation box to reduce the arrival law. Once all QoS adaptation realized, the emulator is a realistic emulation of the real target and the SASM can be validated. A simulation panel is added on the UI (cf. figure 8) and it is then possible to inject fault such as lost and faulty data. In order to reach the final system, we mix the real SAM (UI, mission orders) and the emulated one(robot and camera).

This example illustrates how it is possible to simulate a RTES at different point of its development. Moreover the approach allows flexibility and separation of the concerns between applications behaviors, simulation and target deployment target.

7 Conclusion

This paper presents a MDA-based approach for RTES development. This method gives facilities for early simulation and validation of such systems. The principle is to encapsulate and to separate the information, needed by a system, from the way to produce them (three layers are formally defined). There are several orientations for future works. First one consists in extending tool box for more complex adaptation policies and distributed systems. The second orientation is to provide facilities for the implementation of the proposed models.

References