SAIA, a model driven method for safe deployment of sensors based applications

Julien DeAntoni, Jean-Philippe Babau
CITI laboratory – INSA-Lyon
20, avenue Albert Einstein, 69621 Villeurbanne cedex
julien.deantoni ; jean-philippe.babau@insa-lyon.fr

Abstract

SAIA is a model based architecture for the development of sensors based real time applications. This paper presents the mandatory concepts to manage the extra-functional properties relating to the communication with the physical environment. Moreover, it proposes an implementation of these concepts and then a way to realize a safe application deployment.

1 Introduction

Nowadays, automotive systems have reached a level of complexity which requires to apply both Software Engineering principles and formal techniques. In this context, one major preoccupation is to make an early validation of the key parts of the system. Software is one of these parts. It may be developed and validated using a simulator and then deployed on the real platform without rewriting or tuning it. Moreover, it is interesting to be able to deploy the same software on different platforms and thus reuse the maximum number of validated behaviors.

Based on SAIA (Sensors Actuators Independent Architecture [3, 4]), the paper focuses on one critical aspect of the automotive platforms: the communication part with the physical environment and more specifically the representation of the environment in the system. The main idea of SAIA is to encourage reuse by uncoupling the sensors and the actuators of a specific platform from the inputs and outputs required by an application. The application (mainly a control law) is then based on high abstraction level information (called *Inputs* and *Outputs*) whereas the platform is modeled by low abstraction level drivers information (called *Sensors* and *Actuators*). Then, some complex connectors have to be realized and used to link the platform and the application at the deployment stage.

The deployment is a critical stage of the development. Effectively, a safe behavior depends on the correctness of embedded control laws in software. This correctness is highly related to the Quality of Service (QoS) characteristics of the sensors [7]. The question is then, when reusing a validated application, how to ensure correctness of the system after deployment. For instance, does a behavior validated and tested on a simulator still valid after the deployment on a real platform ?

Concentrating on formal aspects of SAIA and regarding the inputs of a system, the aim of the paper is to propose a formal modelling approach to describe the different QoS parts of the system (application and platform) and then to validate the deployment stage.

After giving the main modelling and validation steps of the approach, the paper presents the chosen QoS semantic. Then techniques to evaluate and to check deployment correctness are presented. To finish, conclusion and future works are presented.

2 SAIA and the QoS

SAIA provides a model driven method and tools for early validation and safe deployment of applications. Focusing on QoS concerns for *Inputs*, SAIA proposes four steps:

1. The formal expression of the QoS for the *Inputs* under which the application behaviors are correct. I.e. the expression of the required QoS.

2. The formal expression of the QoS that characterizes the behaviors of a specific platform (set of *Sensors*). I.e. the expression of the provided QoS.

3. Complex connectors are necessary to connect the platform and the application. These connectors impact and then modify the QoS provided by the platform. Then, the third step consists on an evaluation of the QoS provided by the connectors.

4. The QoS conformity checking. I.e. Does 'the required QoS' 'satisfies' 'the QoS provided by the connectors'

Before to detail each of the previous steps, the next section gives the necessary semantic for QoS description.

3 Expression of the QoS

Due the reactive features of these systems, the physical environment (noted φ) must be represented. The *Inputs* are a view of φ. Classically, an *Input* is represented as a flow of information. A flow d characterises an information as a sequence of occurrences d_i (`i` represents the number from 1 to ∞ of the information occurrence). For each occurrence d_i of an information in the system, we assume that it exists a corresponding physical occurrence φ_i in the environment.

In SAIA, the QoS is a set of QoS characteristics. Each of the QoS characteristics is a QoS-oriented model of a flow. The paper considers the *Inputs* QoS characteristics in the area of real time control systems. Following [9], three QoS characteristics are considered; the two first described temporal characteristic: the `arrival law` and the `delay law`. The third is used only for data : the `accuracy law`. Before to propose a definition of the QoS characteristics, we give the QoS-oriented semantic of the occurrences, noted QoS-occ. In the following definitions, $@d_i/@\varphi_i$ represents the date of the occurrence number `i` of `d`, respectively φ. $V(d_i)/V(\varphi_i)$ represents the value of data d_i, respectively φ_i. The QoS-occ are defined by (see figure 1):

- $arrival_i = @d_{i+1} - @d_i$
- $delay_i = @d_i - @\varphi_i$
- $accuracy_i = |V(\varphi_i) - V(d_i)|$

It exists different ways to describe a QoS characteristic. One of the most simple is an interval. An interval specifies that each QoS-occ belongs to the interval. Thus the interval [1;5] can specify the sequence {1,5,4,3,1,4,2,...} but also {1,1,1,1,...} or {1,5,1,5,1,5,1,5,...} and so on. Unfortunately, some applications need

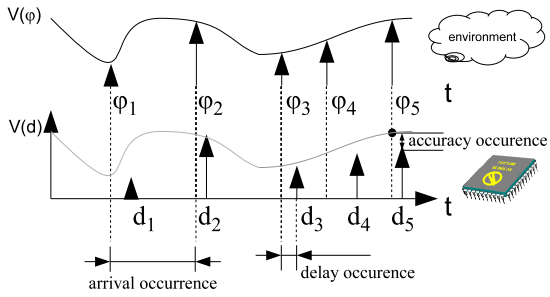

Figure 1. QoS occurrences representation

a more precise specification [7]; for instance, a QoS-occ must belong to [1;5] but as a sequence of unique value. I.e. {1,1,1,...} or {2,2,2,...} ... The intervals are then not expressive enough. We propose to use regular expression to define more precisely each QoS characteristic. A regular expression allow expressing the grammar of a word set. A QoS characteristic is then a set of all the possible word of a regular expression. From the previous examples, the sequence of unique value becomes : $[1^* \,|2^*|\, 3^* \,|4^*|\, 5^*]$ whereas the specification of an interval [1;5] becomes $[1\,|2|\,3\,|4|\,5]^*$.

Because the specification of a QoS characteristic is related to the whole execution of the system, the regular expression must specify a set of infinite words.

3.1 Expression of the required QoS

The required QoS characteristics are specified by the regular expressions that make explicit the flows for which the application realizes the correct behavior. The required QoS is specified for each *Input*. More precisely, for each *Input*, for each QoS characteristic, a regular expression is specified. It is obvious that more restrictive is the regular expression, more difficult it is to satisfy it: to allow a maximum of possible platform to be linked and thus enable a maximum of reuse, the regular expression must represent the biggest set of acceptable words.

3.2 Expression of the provided QoS

The provided QoS represents the QoS which is offered by the drivers of specific sensors. More precisely, for each *Sensor*, the QoS characteristics are evaluated. The evaluation of a provided QoS is not part of this work but exploits the results of [1]. It allows a formal characterization of the sensor drivers depending on their architecture and electronic sensors. The

remainder of this paper considers that for each platform, the provided QoS is known.

3.3 Evaluation of the QoS provided by the connectors

3.3.1 Specification of the connectors

As specified earlier, the application behavior is based on high abstraction level *Inputs* whereas a platform provides low abstraction level information. For instance, the application can specify the '`orientation`' *Input* whereas a platform provides '`right wheel speed`' and '`left wheel speed`'. To construct the '`orientation`' *Input*, aggregation and computation of the platform information are necessary. More generally, to link an application to a platform, SAIA proposes dedicated connectors which are composition of three kinds of basic components: `Format`, `Interpret` and `QoS adapt`.

`Format` is a translation function. Its role is only to change the representation of an information. `Interpret` produces one or more *Input* from one or more *Sensor* information. `QoS adapt` takes charge of explicitly changing the QoS characteristics of a flow (constant delay, periodic arrival law, ...).

`Format` is a function but `Interpret` and `QoS adapt` basic components are composed of two distinct parts:

- a dynamic behavior that specifies the way to collect and produce pieces of information. This part is specified by a timed automaton.
- a function that specifies the way to compute one or more pieces of information (average, unity change, ...).

Following the SAIA component paradigm, the explicit temporal behavior description of a basic component is divided in two parts:

- an time interval whose bounds are the Best Case Execution Time (BCET) and the Worst Case Execution Time (WCET) of the function.
- An additional time in the automaton that represents the possible preemption and blocking time induced by the implementation.

This information is either computed after implementation or given as implementation constraints.

Once the characterization of the components has been realized, a choice must be done concerning the way to aggregate the QoS-occ. When an information aggregation occurs, there are various ways to compute the resulting QoS-occ. For instance, when two pieces of information are aggregated, the resulting delay occurrence can be either the maximum delay of the aggregated pieces of information or the average delay as well as the minimum one. This choice is necessary to compute the QoS provided by the connector. It is often related to the physical dynamic of each information in input (the φ flows dynamic).

We have seen that logical as well as temporal aspect have to be expressed for the specification of the components. For this reason, we chose IF [2], a timed and communicating automata language. The component semantic is then the one of IF: asynchronous communication and discrete time. The QoS evaluation is so based on this semantic.

Due to the connector dynamics (automata), the execution time of the functions (BCET, WCET), the delay induced by the implementation (preemption, ...) and the aggregation strategy (minimum delay, max, ...), the connector realize a non-trivial modification of the QoS flows[3]. The next section outlines the step needed for the quantification of this modification.

3.3.2 QoS evaluation

The evaluation of the QoS provided by the connectors is realized connector by connector. The first step is to isolate the *Sensors* components that interact with the connector we want to evaluate. Then, for each of these components, the second step is the translation of the QoS characteristics expressed as a regular expression into an IF automaton. The IF automaton must produce the information as specified by the regular expression.

This system part can now be executed thanks to the IF toolsuite. To be able to obtain the QoS characteristics, the execution must be monitored. This is done thanks to IF observers. Observers allow to catch and react to events in the system. We propose to use them to measure QoS-occ. The execution of an IF system results in a LTS (Labelled Transition System). A LTS is a graph of states and transitions where states are linked by transition. Each transition contains a label. For each measured QoS-occ, an observer generates a new transition. Then, the resulting LTS represents the combinatorial of every possible execution path where all possible QoS-occ have

been added by the observers, as labels on transition.

To extract the QoS characteristics from the LTS, it just needs to hide all transitions which are not generated by the observers. QoS characteristics are then represented by a LTS. The regular expression can be deduced from this LTS.

The next step to ensure that the platform can be linked to the application is to check if the QoS provided by the connectors satisfies the QoS required by the application.

3.3.3 QoS checking

To check that QoS satisfaction is reached, the "*satisfies*" operator is introduced. Since the required QoS characteristics are the set of acceptable QoS-occ flows, the "*satisfies*" operator must verify that the provided set of QoS-occ flows is only made of acceptable flows. It can be realized differently depending on the way to specify the QoS characteristics.

If a QoS characteristic is specified by an interval, then the "*satisfies*" operator is the inclusion relation ($\subseteq$). Every included intervals produce only acceptable flows because the possible values in the flow are a subset of the required one. If a QoS characteristic is specified by a regular expression, the "*satisfies*" operator becomes a sub-language relation. It means that every possible words in the regular expression of the provided QoS characteristic must be possible in the regular expression of the required QoS characteristic.

Now, to check the validity of the deployment, we have to check two points :

1. the QoS provided by the connector is a sub-language of the required QoS,

2. the QoS provided by the connector specifies only infinite words.

Because the result of the evaluation of the QoS provided by the connector is an LTS, we check these two points thanks to automata analysis. To check the first point, we must verify that all the possible transitions sequences of the provided QoS characteristics LTS exist in the required QoS characteristic LTS: this relation is called 'simulation'. To check the second point, we have to verify that the QoS provided by the connector LTS does not possess final state. These relations can be automatically checked with tools like aldebaran[6].

4 Conclusion

This paper has proposed an approach for the safe deployment of sensors based applications. It identifies the needs in term of QoS and proposes a QoS formalization at three levels: QoS, QoS characteristic and QoS-occ. Then a formal management of these entities is proposed: evaluation and checking. Some related works such as EAST-EEA [8] or REACT [5] projects are of interest. The present work differs from these projects in two points. First it considers the communication with the physical environment: this point is seldom dealt with in the other approaches. The second difference is related to the constraints which are not global constraints such as deadline or deadlock but flow constraints. To finish, an interesting evolution of this work could be the extension of the semantic including probabilistic aspects.

References

[1] B. Ben-Hedia, F. Jumel, and J.-P. Babau. Formal evaluation of quality of service for data acquisition systems. *Forum on specification and Design Language*, 2005.

[2] M. Bozga, S. Graf, and L. Mounier. If-2.0: A validation environment for component-based real time systems. *In ed. Brinksma, K.G. Larsen (Eds) Proceedings of CAV'02*, 2002.

[3] J. DeAntoni and J.-P. Babau. A MDA approach for systems dedicated to process control. *eleventh IEEE International Embedded and Real Time Computing Systems and Applications (RTCSA'05)*, 2005.

[4] J. DeAntoni and J.-P. Babau. Model driven engineering method for SAIA architecture design. *Ingénierie Dirigée par les modèles (IDM'06)*, 2006.

[5] S. Faucou, A.-M. Déplanche, and Y. Trinquet. An adl centric approach for the formal design of real time systems. *In Architecture description language, IFIP*, pages 67–82, 2004.

[6] J.-C. Fernandez. Aldébaran: a tool for verication of communicating processes. *technical report SPECTRE c14, LGIIMAG*, 1989.

[7] F.-L. Lian. Analysis, design, modeling, and control of networked control systems. *Ph.D. Dissertation, The University of Michigan*, 2001.

[8] H. Lönn, T. Saxena, M. Nolin, and M. Törngren. Far east: Modeling an automotive software architecture using the east adl. In *ICSE 2004 workshop on Software Engineering for Automotive Systems (SEAS)*, Edinburgh, May 2004. IEE.

[9] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real Time System*, 1998.