

SAIA: Sensors/Actuators Independent Architecture – a showcase through Martian task specification

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

October 6, 2005

Abstract

Lots of paper deal with the problems of real time systems development. Some of them propose approaches to be independent of the operating system used for the system deployment. This paper introduces a method for development of real time systems independently of the sensors/Actuators (or simulators services) used. The method is explain through the development of Martian task specification. Since it is a simulator-based development, it is well suitable to explain the method and its benefits during the deployment on the real target or when target sensors/actuators changes occurred.

1 Introduction

The goal of Martian Task is to develop a system for an exploration robot. The development of this system is based on a simulator of the robot platform. The matter with these types of approach is that the simulator is not a perfect view of the real platform. Thus, behavior changes can appear during the migration of the system from the simulator to the real platform. Specifically, the simulator rarely provides the same services and the same QoS than a real platform. Then, it is difficult to deploy a system developed thanks to a simulator on the real platform.

In 1976, Edsger Dijkstra described a technique for problem solving [8]:”study in depth an aspect of one’s subject matter in isolation, for the sake of its own consistency, all the time knowing that one is occupying oneself with only one of the aspects”; he dubbed this technique separation of concerns. Nowadays, dealing with the separation of concerns is always a challenge for academic and industrial world. Each domains have their own problems and so their own concerns. Domain such as User Interface or web technology are already applying separation of concerns in a good way [23, 19, 9, 20]. Unfortunately in a real time system such as Martian task, the criticality and the need for hardware specific performance induce a more difficult separation of concerns.

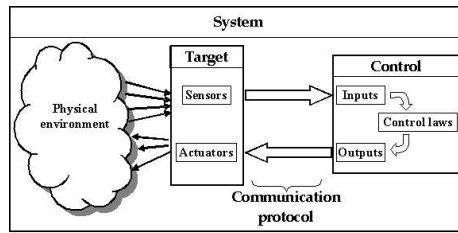


Figure 1: Representation of a SDPC

For an efficient development of Martian Task, it is important to be independent of the operating system as well as to be independent of the environment communication (simulator, sensors/actuators). SAIA [5] is a MDD¹ approach dedicated to Systems Dedicated to Process Control (SDPC). Its goal is to apply separation of concerns between a SDPC and its environment.

A SDPC builds in real time a view of the controlled process (Inputs) by using sensors (Cf. figure 1). Then, it computes the necessary commands (Outputs) in order to control the process. Finally, commands are performed using actuators. Due to their specific and contextual constraints imposed by the environment interaction, SDPC are dedicated systems developed in an ad-hoc and hand-made manner. So, in this type of systems, separation of concerns is null and the resulting specification and code are mainly spaghetti oriented.

This paper presents how the specification of Martian Task can be integrated in the SAIA approach in order to make a separation between the application and its environment communication. The proposed approach and its associated architecture are based on three layers (target layer, control layer and adaptation layer).

After a presentation of the background knowledge and the related work, the SAIA architecture and the associated software engineering concepts are explained. Then, the development process is explained through Martian Task. The implementation mapping is then illustrated. Before to conclude, we evaluate the SAIA method.

It is important to notice that our goal is not to realize a performant control algorithm for robot displacement. The goal is to show how a specific real time application can be specified and realized in order to apply separation of concerns in the communication environment and so, in order to improve easy reuse of software and/or target.

2 background

The Object Oriented (OO) paradigm is widely acknowledged by the industrial world and reinforced by the emergence of a standard (UML [16]). The main interest of OO approaches is their ability to improve the quality of software [11, 21]. But OO approaches lack of architecture structuring capacities and semantic information. To fulfill this gap, MDD propose to use model for system representation. A model, defined by its metamodel, is a system abstraction

¹MDD: Model Driven Development

focusing on a specific concern. Initiated by the OMG ², MDA [14] is proposed for system development in order to increase portability, interoperability and re-usability through architectural separation of concerns. In MDA, the PIM (Platform Independent Model) focuses on the application behaviors and the PM (Platform Model) on the execution environment. A whole system, i.e. a PSM (Platform Specific Model) in MDA terminology, is obtained by models transformation. More precisely, a PSM is obtained thanks to a mapping between a PIM and a PM in MDA terminology. MDA is not a development method but a development philosophy. It gives software engineering concepts and mechanism to use in order to realize a clean separation of concerns between PIM and PSM. Because MDA is a generic method, it does not give information on what are the concerns; they depend on the application domain.

3 Related Work

As state before, concerns separation in MDA sounds differently depending on the application domain. For the majority of the known approach, the goal is to separate the models from their future language implementation[20], [9], [23], [19]. This is a good way to be independent of the execution environment when the operating system and the relying hardware provide all the needed services. It is often done thanks to high level language such as UML [16] or meta-language providing a common API such as[25], [4] in user interface domain. An other way to be operating system independent is the use of a virtual machine [24], [22]. A virtual machine interprets a specific code for the associated operating system. Other approaches, based on middleware [15] allow to be independent of heterogeneous environment and deployment. Unfortunately the problem is more complex when the software correctness relies on the hardware capabilities [7]. To deal with this problem, QoS specifications are needed. From one hand the QoS provided by the hardware and the QoS required by the application as to be described and on the other hand an admission test has to be realized [18]. This specification and admission test has to be done at each level of the used approach [3]. Some approaches adapt virtual machine or middleware to provide QoS management [17][10]. All of these last approaches are focused on the operating system and/or deployment independence. Since real time systems have strong interaction with the external environment, Sensors/Actuators and their associated QoS are important aspects of real time systems but are seldom dealt with. In general purpose systems, Sensors/actuators are always the same and so, can be virtualized [13]. A virtual mouse can then be linked with an electronic paint system, a touch pad, and so on; as long as the associated drivers provide mouse-like services. So, they use a fix interpretation from their hardware to mouse-like services. In real time system, information needed from the environment are more heterogeneous so that it is difficult to virtualize some specific physical device.

The goal of SAIA is to provide an architecture and a methodology to allow a same real time application to be used with different sensors/Actuators target. SAIA is inspired from virtual machine machine and so, introduce an adaptation layer. Of course, to construct a consistent system, the approach introduce a

²Object Management Group

specification and an evaluation of the QoS. The next section details the SAIA method and architecture.

4 SAIA approach

4.1 SAIA concepts

The goal of this approach is to separate the concerns between target, communication protocol and control. The control activity is based on a description of the desired environment view called Input/Output (I/O). It models the state of the environment through data, events and requirements on environment through commands. To use this approach, four questions have to be answered:

1. What Input and Output (I/O) information is useful for the control ?
2. What are the I/O QoS constraints required to ensure the validity of the control ?
3. How can information needed by control (I/O) be produced from sensors/actuators of a specific target ?
4. Is it possible and if yes, how sensors/actuators can provide sufficient QoS ?

The answer to these four questions is given through three models representing the three layers of the system (Cf figure 2). The SAIM (Sensors/Actuators Independent Model) answers to question one and two. Its metamodel is composed of elements characterizing families of Input/Output (I/O). Moreover, this model 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 for the deployment of the system. Then, ALM elements (Adaptation Layer Model) drive us to answer to questions three and four. The fourth question realizes a complex mandatory admission test. All the models are described thanks to a specific architecture described in the next section.

4.2 SAIA architecture

The architecture used in SAIA provides elements giving facilities for modeling of SDPC. The elements associations have to be conform to the metamodel whose a simplified view is given on figure 2 . On this metamodel, data, event and command describe the desired view of the environment (the I/O and the required QoS) while driver data, driver event and driver command describe a platform-specific view of the environment (the sensors/actuators and the QoS provided). The adaptation layer realizes the "smart" link (functional and extra-functional) between these two views. To do that, the adaptation layer is composed by three elements (DAE³, EAE⁴, CAE⁵), each dedicated to a specific type of I/O.

³Data Adaptation Element

⁴Event Adaptation Element

⁵Command Adaptation Element

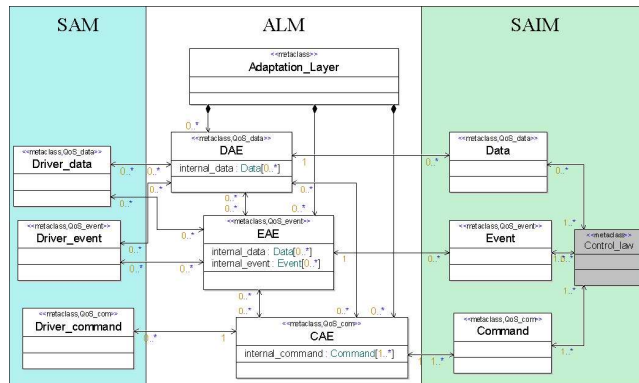


Figure 2: simplified view of SAIA metamodel

5 "Martian Task" Design

In order to respect SAIA methodology, the first question which has to be considered is:

- What Input and Output (I/O) information is useful for the control ?

It is important to notice that the I/O must not make any suspicion about the Sensors/Actuators used by the target (not still chosen). Following Martian task specification, we have to consider what is the information needed to realize the displacement of a robot from one point to another in a broken environment. We can first identify that we need to know the actual position and orientation of the robot as well as its goal position. To know what is the robot doing and so, order the good action, it is important to have a view of the robot state. Because the robot have to evolve in a world, a map of this world is a good help. Moreover, since the environment is broken, we need obstacles information such as the angle of the last obstacle(s) encountered (relatively to the robot) and its(their) type. Then, because the robot can be killed by falling in a hole, the knowledge of an urgent stop is mandatory. Finally, as in all systems, we need to know when the system have to start. Thus we identifies eight Inputs; the specification of these Inputs (related to SAIA) are the following:

- start (event)
- robot_state (categorical data) {normal - crashed - near a hole}
- last_obstacle (type: categorical data and angle: continuous data) respectively {mountain - hole} and $[-\pi; \pi]$
- urgent_stop (event)
- world_view (categorical data) for each point in the space: {nothing - hole - mountain - unknown }
- actual_position (continuous data) X: $[X_{min}; X_{max}]$ and Y: $[Y_{min}; Y_{max}]$
- actual_orientation (continuous data) $[-\pi; \pi]$

- goal_position (categorical data) X: $[X_{min}; X_{max}]$ and Y: $[Y_{min}; Y_{max}]$

Since we need to be able to move the robot, we need a Run and a Turn Output. Moreover when an urgent stop is detected in Input, a stop command is needed in Output. So, we distinguish three Output whose chosen specifications are:

- Run (categorical command) {normal - prudent - scared}
- Turn_of_X (continuous command) $[-2\pi; 2\pi]$
- Stop (boolean command)

Once all I/O have been defined, the control part can be realized. In our implementation the control is a simplistic algorithm: not very clever without history of its movement. But our goal is to illustrate the use of SAIA and not to produce a performant control algorithm. Now that this algorithm is realized, we can treat the next SAIA question:

- What are the I/O QoS constraints required to ensure the validity of the control ?

The major constraint in Martian task is that the robot needs to stop before falling in a hole. It implies that the urgent stop in Input must arrive with a certain delay after what it is too late to stop the robot. Of course, this value is linked to the delay needed to stop the robot once the Output is set. Moreover these two delays depends on the distance of the hole detection and the maximum robot speed. Because at this stage of the development we do not know the Sensors/Actuators used, QoS requirement has to be expressed analytically. The first step consist to make a derivation of constraint between the end to end deadline and: the delay needed to be informed of the urgent_stop, the maximum time induce by the control to react (control WCET) and the delay needed to stop the robot once the Output is set. The end to end deadline is equal to the distance of the hole detection divided by the maximum speed of the robot. During constraints derivation, we consider that the delay required to be informed of an urgent_stop is the same as the delay required to stop the robot. thus we obtain that:

- $end_to_end_deadline = \frac{hole_detection_length}{max_robot_speed}$
- $urgent_stop.delay_{max} = \frac{end_to_end_deadline - WCET(control)}{2}$
- $stop.delay_{max} = \frac{end_to_end_deadline - WCET(control)}{2}$

Now, the Sensor Actuator Independent Model (SAIM) is completed. In the next step of the development, the target is choice. Here, it is the Martian task simulator. Thus, we can instantiate the constraint and we obtain:

- $end_to_end_deadline = \frac{0.75m/s}{2m/s} = 375ms$
- $urgent_stop.delay_{max} = \frac{375ms - 15ms}{2} = 180ms$
- $stop.delay_{max} = \frac{375ms - 15ms}{2} = 180ms$

We can thus deal with the next question of SAIA methodology:

- How can information needed by control (I/O) be produced from sensors/actuators of a specific target ?

Even if the simulator provides about high level information (bumpers and sonars sensors are already aggregated, the state of the robot is given, and so on); two operations must be realized to answer this question: `format()` and `interpret()`.

`Format` is in charge of the formatting needed to realized consistent interpretation. For example, the orientation of the robot provided by the simulator is given in the form $[-\infty; \infty]$ but the corresponding Input (actual_orientation) is specified as $[-\pi; \pi]$.

Once the formatting is realized, `interpret()` method realizes the semantic change which can be need to produce the Input from drivers information. For instance in Martian task, we need to interpret: the state of the robot, the position and the orientation of the robot and the bumper and sonar information to produce a map of the world. An other interpretation example is the production of the `urgent_stop` event thanks to the sonars information. The figure 3 is a graphical representation of what Sensors/Actuators (simulators services) are used to produce Inputs/Outputs through the adaptation elements. It represents the whole Martian task architecture.

Each simulator services have a QoS provided. Because we realize a formatting, an interpretation and data aggregation, this QoS is modified by the adaptation layer. However, the resulting QoS can be calculated thanks to the tool box provided by SAIA. When the resulting QoS provided has been calculated, the admission test has to be done. In an other word, we have to deal with the next question:

- Is it possible and if yes, how sensors/actuators can provide sufficient QoS ?

In the Martian case, there is no need to realize QoS adaptation such as interpolation or filtering to reach the sufficient QoS. What ever, the QoS adaptation described in SAIA can be used to degradate the QoS provide by the simulator and so be closer to the QoS provided by a real platform (see section 7).

The resulting model is called SASM for Sensors/Actuators Specific Model. It is a model independent of the operating system but specific to the Sensors/Actuators. The figure 3 gives a view of this resulting model. This model has been realized in GME, a Generic Modeling Environment customized thanks to SAIA metamodel. The next section give an overview of this SASM implementation.

6 Martian Task SASM implementation

Now that the SASM is realized, it has to be implemented in a specific language and mapped on a task model [6]. There are different strategy to implement models. In object-based modeling, for real time systems, it is necessary to provide concurrency management of real time objects. Two approaches (known as "object-based" and "event-based" management) are applied for this purpose [2]. Object-based management is simpler but increase critical section and CPU

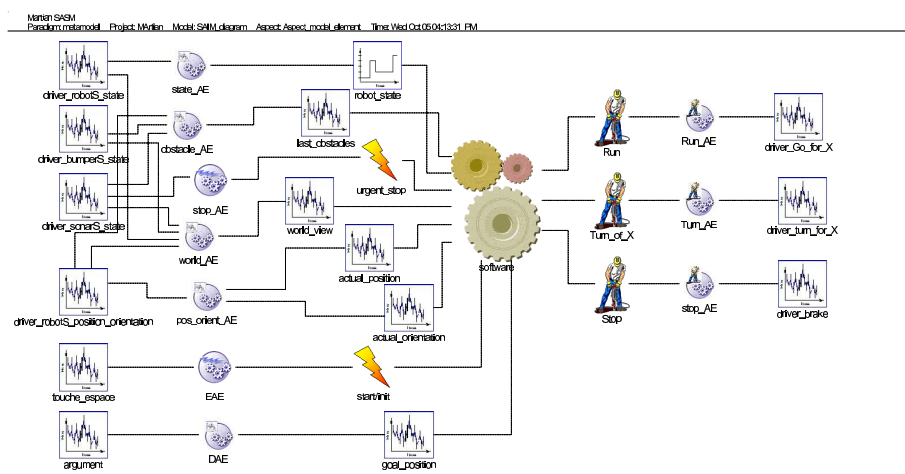


Figure 3: SAIA model of Martian task system

overload. Event-based [12], [1] management allows reducing blocking time, so "event-based" approach was chosen. In this approach concurrency is described as orthogonal to the objects and is linked to external events. Its first step is to list the event that trigger a real time activity. In Martian task, the only real-time activity trigger event is a sonar detection. An "event thread" is then associated with this event. This thread is characterized by the call sequence of the operations responding to the event. Objects linked to the event thread are then grouped together. Since some sensors (here simulator services) information is involved in the call sequence of the event thread, sensors (simulator services) reading has been grouped into a specific thread. Thus, we distinguish three threads (and three groups):

- The periodic sensors reading
- The real time event thread
- A periodic thread regrouping the normal activities

The period of the sensors reading impacts on the urgent stop detection delay whereas the period of the normal activities impacts on the trajectory quality (more or less jerky) and on the CPU load. It is important to notice that certain objects appear in more than one group. Suitable protections are therefore required. We used mutex semaphore to ensure mutually exclusive access to each object attribute. Once the concurrency strategy has been selected, a priority has to be given to each thread. Since the event thread uses sensors (simulator services) information, the maximum priority is given to the periodic sensors reading. The next maximum priority is then given to the event thread and of course, the lowest priority is attributed to the periodic thread regrouping the normal activities.

7 SAIA evaluation

It is clearly explain in the previous sections that the control is not based on Sensors/Actuators (or simulator services) but on high level Inputs/Outputs descrip-

tion. It is also show how the link between these I/O and the Sensors/Actuators is realized by the adaptation elements. This section intents to explain what are the benefits introduced by the method for the development of real time systems.

Even if simulators are a good help during the development of complete systems, they always lack of realistic services in term of behaviors as well as in term of QoS. For instance in the Martian Task simulator, the reading of the sonar is disturbed by probabilistic false positive and false negative. The matter is that when a false negative detection occurred, successive reading give the same false positive as long as the robot has not moved or turned. This type of comportment is not the same as a real sensors one. An other example of unreal thing is that the simulator brake instantaneously. A real robot has at least a little inertia which implies a delay before the complete stop of the robot. Moreover it is possible that the real target does not give the same services as the simulator by giving, for instance, the position relatively to the start point or the sonars/bumpers detection individually for each physical sensors. It is than difficult to improve the control of real time systems in these conditions.

To deal with these problems, SAIA approach allows to test the control (SAIM) with more realistic performance by adding delay or introducing loss on the Sensors/Actuators information. This is a mandatory step if the control (SAIM) must be validated for realistic Sensors/Actuators.

Moreover, once the SAIM is validated for a specific target, the approach makes changes occurring in Sensors/Actuators easier. Effectively, because formatting and interpretation of Sensors/Actuators are separated from the control, the changes occur locally to a method in the adaptation element and is not weaved between several objects. For example, in Martian task, the position is given from triangularization; on a different platform, it could be given from the start position and the number of wheel turn. In this case, only the `format()` and the `interpret()` method of the associated adaptation element have to be changed. Moreover, the tool boxes provided by SAIA helps evaluating the resulting QoS calculi according to the aggregation semantic. Thus for a deployment from a simulator-based to a real platform as well as for a change in Sensors/Actuators information, the approach use the same SAIM without any changes.

8 Conclusion

This paper presents the development of Martian Task thanks to SAIA approach. 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 formally defined layers are used). there are several orientation for future works on SAIA method. The first one is to give facilities for implementation of the proposed model. The second orientation consists in extending tool boxes provided by SAIA for distributed systems.

References

- [1] A Moore. Systems development - from conception to delivery. *Embedded systems programming europe*, 1999.

- [2] A Muth and T Kolloch and T Mier-Komor and G Frber. An evaluation of code generation strategies targeting hardware for rapid prototyping of sdl specifications. *IEEE Workshop on Rapid System Prototyping*, 2000.
- [3] ARTIST. Adaptive real-time systems for quality of service management. *Draft IST-2001-34820*, 2003.
- [4] O. S. Associates. Openui v4.0. <http://www.osa.com.au>.
- [5] J. D. J.-P. Babau. A mda-based approach for real time embedded systems simulation. *Distributed Simulation of Real Time and embedded system*, 2005.
- [6] J.-P. Babau and S. Gerard. *Model-Driven Schedulability Analysis*, chapter 9. Hermes Science publishing, 2005.
- [7] B.Selic. Physical programming: Beyond mere logic. *EMSOFT 2002, LNCS 2491*, 2002.
- [8] E. W. Dijkstra. *The characterization of semantics*, chapter 3. Prentice-Hall, 1976.
- [9] D. Frankel and J.Parodi. Using model-driven architecture(tm) to develop web services. *IONA Technologies PLC, White paper*, 2002.
- [10] java community process. Real-time specification for java. <http://jcp.org/en/jsr/detail?id=1>, 2002.
- [11] J.Rumbaugh, M.Blaha, W.Premierlani, F.Eddy, and W.Lorensen. Object-oriented modeling and design. *Prentice-Hall*, 1991.
- [12] M Awad and J Kuusela and J Ziegler. Object-oriented technology for real time systems. *Prentice Hall*, 1996.
- [13] Microsoft. Review of windows 3.1 architecture. [http : //www.microsoft.com/technet/archive/wfw/1ch2.msp](http://www.microsoft.com/technet/archive/wfw/1ch2.msp), 2005.
- [14] OMG. Model driven architecture guide v1.0.1. <http://www.omg.org/mda>, 2003.
- [15] OMG. Corba 3.0.3 specification. [http : //www.omg.org/technology/documents/formal/corba_iop.htm](http://www.omg.org/technology/documents/formal/corba_iop.htm), 2004.
- [16] OMG. Final ftf report mof 2.0 core and uml 2.0 infrastructure finalization task force. <http://www.omg.org>, 2004.
- [17] OMG. Real time corba specification. [http : //www.omg.org/technology/documents/formal/RT_dynamic.htm](http://www.omg.org/technology/documents/formal/RT_dynamic.htm), 2004.
- [18] OMG. UML profile for modeling QoS and fault tolerance characteristics and mechanisms. *OMG adopted specification*, 2004.
- [19] Oscar Nierstrasz and Franz Achermann. Separation of concerns through unification of concepts. *ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, 2000.
- [20] P.Caceres, E.Marcos, and B.Vela. A MDA-based approach for web information system development. *wisme*, 2003.
- [21] P.Coad and E.Yourdon. Object-oriented analysis. *Prentice-Hall*, 1991.
- [22] SmallTalk. <http://www.smalltalk.org>.
- [23] Sofie Goderis and Dirk Deridder. A declarative dsl approach to ui specification - making ui's programming language independent. *Workshop on Evolution and Reuse of Language Specifications for DSLs (ERLS)*, 2004.
- [24] sun. Java 2 platform standard edition development kit 5.0. <http://www.java.sun.com>.
- [25] C. M. University. Amulet v2.0. [http : //www.cs.cmu.edu/ amulet/amulet - home.html](http://www.cs.cmu.edu/amulet/amulet-home.html).