

CiberMouse design: a case study for SAIA model reuse

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

In the context of last year RTSS robotic competition, SAIA models and their implementation have been proposed. The goal was the development of an exploration robot. This year, the paper proposes the reuse one of these models for the ciberMouse design. The realized modifications are presented and an emphasis is made on benefits and limits of SAIA model reuse.

1 Introduction

CiberMouse is the second robotic competition holds in conjunction with the Real Time System Symposium (RTSS) conference. Its goal is the development of an efficient real time software for an exploration robot. The development is based on a simulator of the real robot platform.

Last year, the competition was the maRTian project. The goal of the maRTian robot was to go from one spot to another one with a minimum knowledge about its environment. CiberMouse presents similarities with the maRTian project, the robot must go from one spot to another one (the beacon) and come back to the starting spot. The difference is that the beacon position to reach is not a priori known. Moreover, the services offered by the simulators this year and last year are almost different.

Based on SAIA (Sensors Actuators Independent Architecture) [2], models and implementation have been realized for the maRTian project [1]. SAIA is an architecture for the development of real time software independently of a specific platform sensors and actuators (or simulator). It proposes a first model independent of the platform services. Then it is linked thanks to a complex connector to the platform services. [1] is a showcase of SAIA and highlight the benefits of a such approach during the deployment on the real platform.

The development of ciberMouse is also SAIA based.

Moreover, because similarities exist between ciberMouse and maRTian, the question addressed here is the one of the reuse: is it possible to reuse maRTian models and implementations for the ciberMouse design ? This way, it allows evaluating SAIA approach for reuse.

The goal of this paper is to present how the first model and its implementation, realized in the scope of the maRTian competition, can be reuse for the ciberMouse design. The first section quickly presents SAIA and the choice done for the maRTian models. The second section explains how the last year model has been reused. Then, we evaluate the reuse, its benefits and its limits.

2 SAIA and the maRTian models

SAIA is based on three models (see figure 1). The first one (called SAIM: Sensors Actuators Independent Model) express the behaviors of the application without any suspicion about the sensors and actuators or simulator services. The application is based on high abstraction level *Inputs* and *Outputs*. The second model (called SAM: Sensors Actuators Model) represents the sensors and actuators drivers services or the corresponding simulator services. Then the third model (called ALM) is the specification of the complex link between the first and the second model. The ALM is composed of two kinds of adaptation element. The *Input* adaptation elements are in charge of the *Inputs* construction from sensors (simulator) data and event. The *Output* adaptation elements are in charge of the *Outputs* realization thanks to actuators (simulator) command. Each adaptation element is composed of three basic components: **Format** is a translation component. Its role is only to change the representation of an information. **Interpret** produces one or more *Input* from one or more *Sensor* information for *Input* adaptation element and produces one or more *Actuators* information from one or more *Output* for *Output* adaptation element. **QoS adapt** takes charge of explic-

itly changing the QoS characteristics of a flow (constant delay, periodic arrival law, ...).

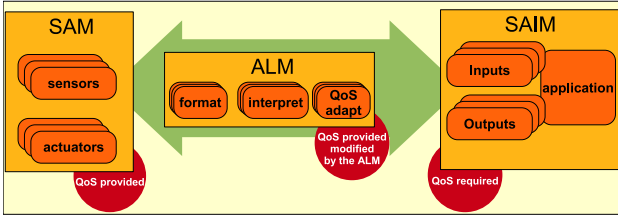


Figure 1. SAIA models

In maRTian as well as ciberMouse, the SAM are given by the simulators interfaces. They are almost different so that no reuse is possible at this level. However, mission goals encapsulated in the SAIM presents similarities. Then we propose to reuse the maRTian SAIM and so, it is the only detailed model. To realize the maRTian SAIM model, eight *Inputs* (in which four consigns) and three *Outputs* have been defined:

Inputs

- **start** (event consign)
- **robot_state** (categorical consign) {normal - crashed - near a danger}
- **goal_position** (categorical consign) X:[X_{min} ; X_{max}] and Y:[Y_{min} ; Y_{max}]
- **urgent_stop** (event consign)
- **last_obstacles** (continous data) [$-\pi$; π]
- **world_view** (categorical data) for each point in the space: {nothing - hole - mountain - unknow }
- **actual_position** (continous data) X:[X_{min} ; X_{max}] and Y:[Y_{min} ; Y_{max}]
- **actual_orientation** (continous data) [$-\pi$; π]

Outputs

- **run** (categorical command) { normal - prudent - scared}
- **turn_of_X** (continous command) [$-\pi$; π]
- **stop** (boolean command)

The application that uses these *Inputs* and the *Outputs* realizes a path finder with obstacles avoidance. This model and its implementation have been validated on the simulator by adding delay and changing arrival laws of the *Input* update and *Output* performing. It results in a SAIM model whose arrival law and delay for correct behaviors are known. The goal is now to see how this model and its implementation can be reused.

3 the maRTian models for ciberMouse

To realize its mission, the main functionality of maRTian is a path finder. It is obvious that ciberMouse needs to use a path finder at least to go back to its initial position but it also needs other features such as: looking for the beacon, moving to the beacon and giving mission state information to the simulator (by using leds). Enforcing reuse, the new models has to incorporate these features, which are not in the maRTian project, with a minus change on the validated SAIM model and its implementation. Since the SAM is provided and can not be modified, the changes will appear in the ALM model, i.e. in the way to link the SAM and the SAIM.

We are first going to detail the changes that appear in the SAIM before to detail the others major modifications, realized in the ALM.

3.1 SAIM modifications

The maRTian SAIM has been modified; more exactly it has been extended. The first extension is related to the mission state information. By looking the *Inputs* and *Outputs* of the maRTian SAIM, we founded that no *Input* or *Output* can be used to drive the mission state information for the simulator. So, we add an *Output* called **mission_info** which informs about the mission state. Three different mission states are of importance in ciberMouse: the robot is on the beacon, the robot returns to its initial position and the robot has finished.

The second extension concerns the way to drive the **mission_info** *Output*. It consists in the add of two items in the **robot_state** *Input* consign: **on_target** and **on_end**.

It is the only changes done on the maRTian SAIM, The fonctionnality realized by this model is always a path finder. The next section presents how to integrate the other features needed by ciberMouse in the system.

3.2 ALM modifications

There are two things the ALM must do. First, it must use the SAM services to create the SAIM *Inputs* and to perform the SAIM *Outputs*. Secondly, the ALM must 'drive' the SAIM in order to realize the features which are not related to the path finder. Each of these two ALM objectives are presented through:

- the way to provided the robot **actual_position** while the SAM does not provide it

- the way to produce the `goal_position` to fulfill all the ciberMouse mission features.

The absolute robot position (`actual_position`) is mandatory for the application correctness. To provide it, an open loop is realized (see figure 2). First, the

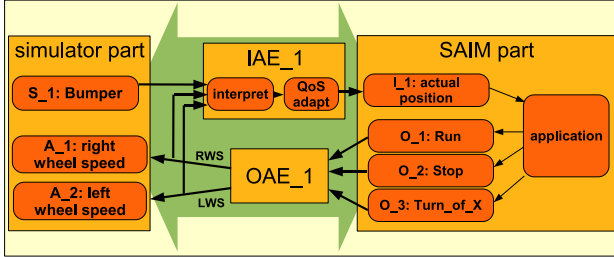


Figure 2. Actual position computation

moving *Outputs* (O_1 O_2 and O_3) are translated by OAE_1 into two low level commands LWS and RWS (Left/Right Wheel Speed) for A_1 and A_2 in the simulator. LWS and RWS are also send to IAE_1 by OAE_1. Effectively IAE_1 is in charge of production of the `actual_position Input`. With LWS, RWS and a representation of the motors dynamic (provided in the ciberMouse specification), a computation of the robot position is possible. So, LWS and RWS are handled in IAE_1 by an *Interpret* component. This component contains the motors dynamic and must aggregate consequently LWS and RWS to evaluate the robot position. The environment is represented by the simulator; and so its evolution is discrete. The computation is then realized for each simulator cycle to have a maximum accuracy. Then, because the update of the `actual_position` is not usefull at each cycle, a *QoS_adapt* component acts as a filter and update the *Input* only when needed (once on three in our case). If the overhead introduced by the position computation is too big, it is possible to compute it less often; but to avoid a too big drift, all the low level motors command must be taken into account.

To finish, we have to consider the physical constraints. The robot can not move forward if there is a wall or an other robot in front of him. The bumper reflects this information and so the bumper sensor S_1 is also linked IAE_1. When the bumper is 'on', the position does not change.

Contradictly to the maRTian project, the first spot to reach is not known and depends on the mission state. It signifies that the absolute position `goal_position` can not be known at the begining of the execution. There are three mission states:

- the research of the beacon,

- the move to reach the beacon,
- the return phase.

For each mission state correspond a kind of `goal_position`. The '`goal_position Input` adaptation' element takes charge to their computation. More precisely, for each mission state, there is a different behavior:

- initially, it must set the `goal_position` thanks to a map exploration policy to have a chance to see the beacon at least one time;
- when the beacon have been seen, it must set the `goal_position` to catch up with the beacon;
- then, it must set the `goal_position` to the initial position when the robot is in the return phase.

To fulfill these requirements, the '`goal_position Input` adaptation element' must be linked to: the beacon *Sensor*, and the '*Inputs* adaptation element' in charge of: `actual_position`, `actual_orientation`, and `robot_state`. However, depending on the mission state, the elements which are linked to the adaptation element are necessary or not.

Despite that the ALM has been almost totally modified, the SAIM has been reused with a minimum of modifications. The next section discusses this reuse and its impacts on the system.

4 Reuse analysis

A first work we have done (not presented here) was to deploy the maRTian SAIM with the exact maRTian behaviors on the ciberMouse platform (SAM). This experience has demonstrated the ability to reuse a SAIM on different platform. In this case the modifications have occurred only in the ALM.

Then, the goal was to reuse the maRTian SAIM in a different context; in term of platform as well as different use cases (following the ciberMouse rules). To achieve that, a SAIM modification was mandatory: the add of the *Output user_infos* and the way to drive it. Even if the maRTian project had no requirements for that, it was suprising to consider an embedded system without any way to communicate its state to the environment. To avoid future modification and improve the SAIM reusability, the *user_infos Output* is modified in such way that it notifies all kinds of mission state, not only the one of importance in ciberMouse. For instance, it contains these mission states: `wait_start`, `look_for_beacon`, `avoid_obstacle`,

path_finder_to_X_Y, and so on. After these modifications in the SAIM, it is always possible to use it for the maRTian project. The SAIM modifications can then be seen as an improvement of the first maRTian SAIM. It signifies that the job skills are better encapsulated after some reuse of the same model and so provide a more generic "exploration robot" SAIM.

The management of the `goal_position` *Input* outlines that the ALM is more complex than in the maRTian project. It is due to the encapsulation of some mission features into the ALM rather than into the SAIM. To avoid a too complex ALM, two solutions have to be considered. The first one consists in slicing the application in such way that the different functionalities and the life cycle are all isolated in different modules. This way, a functionality can easily be added. However, this solution is outside of SAIA scope.

The other solution consists in realizing a sequence of different SAIM (see figure 3). In this case, a SAIM uses *Inputs* produced from *Sensors* to produce *Outputs* (as the previously presented SAIM). Then, a second SAIM considers the *Outputs* of the first SAIM as *Sensors*. Consequently it uses them to produce new *Inputs* and so, new *Outputs*. This solution is specifically adapted when the adaptation behaviors realized by the first SAIM needs to be reused.

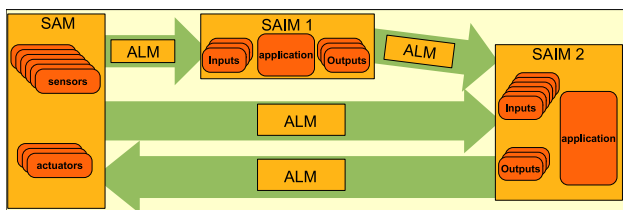


Figure 3. An instance of a SAIM sequence

All that precedes is driven by a strong will to reuse the maRTian model. Without this will, the ciberMouse models would be driven only by the ciberMouse specification and so would be different. For instance, there are three kinds of `goal_position`, each representing a different knowledge depending on the mission state. These three different knowledges would have been dissociated into two *Inputs* for this hypothetical SAIM:

- `goal_position`: which would be the calculated position relatively to the robot position
- `initial_position`: the absolute position to return to.

Following the SAIA paradigm, the position to reach when looking for the beacon is not an *Input* because it is not calculated from an environment knowledge but rather from an exploration policy. This exploration strategy would be encapsulated in the SAIM.

Then, the resulting ciberMouse specific SAIM is not only a path finder. It also contains an exploration strategy and a more complex life cycle. Because more functionalities are encapsulated in the SAIM, it results in a more simple ALM. When a platform change occurs, the ALM is widely impacted. So more simple the ALM is, easier the platform change is. Contradictly, more simple the SAIM is, easier it is to use it for another specification. It outlines that the way to reuse a model depends on the future purpose of the re-used model.

From a QoS point of view, the QoS can be expressed in different ways. For instance, the QoS delay is not the same for the `goal_position` to reach when the beacon has not been seen and for the `goal_position` when the robot comes back. With SAIA it can not be expressed for a same *Input*. It signifies that the reuse is possible and accelerates the development but it also limits the QoS expressivity.

5 Conclusion

This paper outlines the reuse opportunity when using SAIA models. It focuses on the reuse of the SAIM model. The introduction of high level *Inputs* and *Outputs* creates a well specified interface between the application and the platform and thus, facilitates the platform change. The behaviors expected by the application newly developed (ciberMouse) differs in some point from the original application (maRTian). The additional features are encapsulated in the ALM which becomes more complex. By realizing a sequence of SAIM, the reuse seems to be better and the ALM more simple. It could be interesting to focus on this alternative in the offing. On a longer view, it could be interesting to study how the QoS specification can be linked to an application life cycle

References

- [1] Julien DeAntoni and Jean-Philippe Babau. SAIA: Sensors/Actuators Independent Architecture - a showcase through martian task specification. *Proceedings of the ERTSI 2005 - Embedded Real Time Systems Implementation Workshop*, pages 43–50, 2005. <http://www.cs.york.ac.uk/ftplib/reports/YCS-2005-397.pdf>.
- [2] Julien DeAntoni and Jean-Philippe Babau. Model driven engineering method for SAIA architecture design. *Ingénierie Dirigée par les modèles (IDM'06)*, 2006.