Available online at www.sciencedirect.com
ScienceDirect
Engineering Applications of Artificial Intelligence 21 (2008) 805–823
Engineering Applications of ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

Agent-based distributed architecture for mobile robot control

J.L. Posadas*, J.L. Poza, J.E. Simó, G. Benet, F. Blanes

Departamento de Informática de Sistemas y Computadores (DISCA), Escuela Técnica Superior de Informática Aplicada (ETSIAp), Instituto de Automática e Informática Industrial (AI2), Universidad Politécnica de Valencia, Camino de Vera, s/n. 46022 Valencia, Spain

Received 22 February 2007; received in revised form 19 June 2007; accepted 16 July 2007
Available online 7 September 2007

Abstract

Mobile robots are physical agents that move and interact continuously while embedded in a dynamic environment. Communications can be one of the most difficult parts of building robot architecture because of the increasing complexity of sensor and actuator hardware, and the interaction between intelligent features and real-time constraints. Currently, hybrid architectures offer the most widespread solutions for controlling intelligent mobile robots. This paper deals with the communications framework necessary to design and implement these architectures. The main goal of this work[1] is to design a modular and portable architecture that allows the development of robot control systems. A multi-level and distributed architecture based on the reactive/deliberative paradigm is presented. Its main components are mobile software agents that interact through a distributed blackboard communications framework. These agents can be run on onboard processors, as well as on fixed workstations depending on their real-time restrictions. The presented control architecture has been tested in a real mobile robot and results demonstrate the effectiveness of distributing software agents to guarantee hard real-time execution.

Keywords: Mobile robots; Distributed computer control systems; Object-oriented technology; Agent-based systems; Real-time systems

1. Introduction

Robots usually have different kinds of sensors distributed in various nodes. Sensory information is used to calculate actions that the robot will execute. In some architectures, robots behave reactively (Brooks, 1986) by making couplings between sensor values and actuator actions, and these couplings are known as behavioural patterns. Sensory information is also used to obtain a symbolic internal model of the environment. This model enables planning and prediction processes to be made that work by considering the system's current information and past information. This behaviour is known as deliberative behaviour (Nilsson, 1980).

Current trends in mobile robot control architectures (Orebäck and Christensen, 2003; Janglova et al., 1996; Janglova and Vagner, 1999) are based on both models of behaviour—reactive and deliberative. This combination is known as hybrid architecture (Arkin, 1990) and has two levels: reactive and deliberative. The deliberative level has to obtain and offer the reactive level those behavioural patterns that are necessary for the robot to achieve its objectives. The reactive level has to execute these behavioural patterns by guaranteeing real-time restrictions. The communications framework is the base that enables the necessary interaction between reactive and deliberative levels, by sending distributed sensory information to tasks at both levels and sending actions to actuators.

Deliberative and reactive tasks can be structured in a natural way by means of independent software components. Multilevel hybrid architecture design is based on the modularity of software and the independent execution of components (Murphy, 2000). Behavioural patterns obtained from the deliberative level and executed in the reactive level are independent processes which are executed concurrently. The implementation of behavioural patterns can be based on *schema theory* (Arbib, 1986; Arkin, 1998). According to this theory, a behaviour (Fig. 1) is composed

*Corresponding author. Fax: +34 96 387 75 79.
E-mail addresses: jposadas@disca.upv.es (J.L. Posadas), jopolu@disca.upv.es (J.L. Poza), jsimo@disca.upv.es (J.E. Simó), gbenet@disca.upv.es (G. Benet), pblanes@disca.upv.es (F. Blanes).
[1]This work has been partially funded with Spanish government project DPI2005-09327-C02-01/02.

0952-1976/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2007.07.008

of at least one *motor schema* and at least one *perceptual schema*.

The architecture design, suitable for hybrid systems, has to support the distributed nature of data sensing and behaviour. The main idea is that the communications framework will be the basis for constructing the motor and perceptual schemas and must be extendable to the deliberative level of computing.

The proposed architecture enables the system engineer to express the application logic in two levels of intelligent behaviour: deliberative and temporal reasoning, and emergent intelligence by the interaction of simple reactive behaviours.

The paper is structured as follows. Section 2 describes related work. Section 3 presents a proposed architecture overview. Sections 4–6 detail communications framework, reactive and deliberative levels of the architecture. Section 7 describes an implementation example and results. Finally, Section 8 summarizes the conclusions.

2. Related work

Significant hybrid architectures (Arkin, 1998; Murphy, 2000; Bonasso et al., 1997; Konolige and Myers, 1998) are presented by describing their components, multilevel organization, and the behaviour execution mechanism. However, communications infrastructures are not specified in architecture models. The communications infrastructure specification is necessary because of its impact in organizational and execution models of the architecture. The communication infrastructure establishes how architectural levels interact, how architectural components communicate, how they access distributed data, and how behaviours are executed.

Currently, the importance of communications infrastructure is growing. Different groups (Brooks et al., 2005; Schlegel, 2006) are taking into account the use of communications patterns as a key to component-based robotics. They mainly agree with the following communications requirements, a more complete description of which can be found in (Schlegel, 2006):

- Components must interact asynchronously with each other via a predefined communication interface.
- Communication objects must be transmitted *by value*.
- Components must be modular and re-useable.

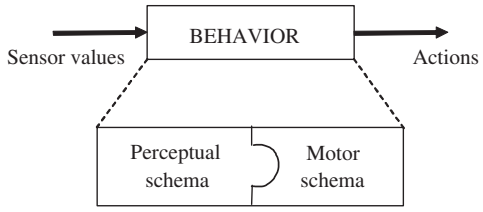

Fig. 1. Behavioural patterns according to schema theory.

- Components must be able to be dynamically wired at runtime.
- Communications frameworks must hide all the communications and synchronization issues.
- Communications frameworks must provide predictability of the time needed for communication.

Also, from our point of view, communications framework design must take into account the various temporal constraints between reactive and deliberative levels of hybrid architectures. The reactive level executes tasks which must respond to the changes in the environment. These tasks have associated hard real-time restrictions, for example, a task that avoids obstacles must react to proximity sensor signals in a time limit determined by the speed of the robot. Traditionally, hard real-time execution is resolved by previously planning the execution of the processes. However, new models are required to resolve hard real-time execution (Brennan et al., 2002) in a system where environmental conditions change dynamically and processes are not known a priori (or open systems, such as when a mobile robot moves in an unknown environment by executing unpredictable behaviours). The solution tends to define systems which adapt dynamically by delegating the execution of non-critical tasks to other nodes, or by degrading their performance, and increasing execution time periods (Hassan et al., 2001). The deliberative level executes tasks which have soft real-time restrictions. When a deliberative task does not achieve its execution period, the system is not exposed to a risk of catastrophic malfunction. Deliberative tasks must execute spatial–temporal fusion processes in order to build a symbolic internal model of the environment by using the sensor values. A communications framework must be able to provide all the data associated with its temporal information. Therefore, tasks can validate useful data by using their time properties. The calculation of temporal information tends to use temporal firewall mechanisms (Kopetz and Nossal, 1997). This approach is based on adding time delays by accumulating all communication overloads.

Currently, there are architectures that follow these patterns, such as GenoM/Kheops (Alami et al., 1998), 4D/RCS (Albus, 2002), CoRoBa (Colon et al., 2006) or ORCA (Brooks et al., 2007):

- GenoM (Alami et al., 1998) is a tool that helps build real-time software architectures. The general architecture is composed of a decision level and a functional execution level. Kheops is a tool for the reactive control of the functional level. Before running, a knowledge base consisting of propositional rules is necessary so that Kheops can guarantee real-time execution.
- 4D/RCS (Albus, 2002) consists of a multi-layered multi-resolutional hierarchy of computational nodes each containing elements of sensory, processing, word modeling, value judgment, and behaviour generation. At the lower levels, these elements generate goal-seeking

reactive behaviour. At higher levels, they enable goal-defining deliberative behaviour. They use Neutral Messaging Language (NML) (Gowdy, 2000) as communication middleware. NML is designed to be reconfigurable but not at run-time. NML cannot react to incoming data, so clients have to poll for incoming data.

- CoRoBa (Colon et al., 2006) is a multi mobile robot control framework based on software reusability. It uses CORBA as its communication middleware and, consequently, does not target hard real-time applications. The CORBA component model lacks sufficiently mature implementations (Brooks et al., 2005).
- ORCA (Brooks et al., 2007) is an open-source Component-Based Software Engineering (CBSE) framework designed for mobile robotics. An ORCA component is a stand-alone process. It interacts with other components over a set of well-defined interfaces.

Our architecture proposal tries to merge the strong points of the architectures mentioned above while minimizing their weak points. It presents an original, simple, and effective solution based on the use of mobile software agents that interact through a communications framework that uses different connection buses to guarantee temporal constraints.

2.1. Distributed communication models

There is a connection between distributed programming models and distributed communications models (Brugali and Fayad, 2002). It is necessary to communicate between different software components or activities. As communications models have evolved they have produced dynamic systems whose main characteristics are: communication regardless of activity location; data mobility (in fact, activity reference mobility similar to CORBA or Java RMI); activity mobility (in fact, an activity code mobility similar to Java RMI or dot NET); and recently, reflection mechanisms (like Java or dot NET).

All these characteristics are included in multi-agent systems (MAS). Software agents represent a natural way of modeling and implementing complex, open, and distributed systems (Jennings and Wooldridge, 1998). Modularity and independence are innate characteristics of software agents. Mobile agents (Lange and Oshima, 1999) can reduce network load and avoid network latency. They can also dynamically adapt to the environment. As a result, software agents can be useful for designing and implementing hybrid architectures to control mobile robots.

3. SC-Agent: architecture proposal

The proposed architecture is called SC-Agent and is based on blackboard technology. Some preliminary architectural aspects were presented in Posadas et al. (2004). The support of blackboard architecture gives to the engineer a very expressive framework to develop intelligent and inference-based agents. Moreover, the explicit inclusion of temporal information enables agents to do reasoning about time beyond the hidden automatic load balancing provided by the agent's migration mechanism. This proposal intends to achieve the following objectives:

1. To configure and control distributed industrial systems in general, and mobile robot systems in particular, by means of a hybrid reactive/deliberative structure.
2. Modularity and reusability based on software agents.
3. Adaptability and portability based on the mobility of the agents.
4. Communications framework that considers differing temporal constraints between tasks and allows the separation between communication time and processing time. Initially, the required code is moved to hosts where it is subsequently executed locally.

SC-Agent architecture (Fig. 2) is a multi-level and hybrid architecture that is composed of three distributed parts: a deliberative level, a reactive level, and a communications framework. Its main components are mobile software agents. Relations and interactions between components are performed through the communications framework.

The deliberative level is a soft real-time system that represents a higher level of knowledge. It executes planning tasks (mission planning agent) based on the state of a symbolic internal model of the environment. As a result of these planning tasks, mission planning divides objectives into behavioural patterns (perceptual and motor schemas) which are sent to the reactive level.

The reactive level is a hard real-time system which is connected using sensors and actuators. This level reacts to sensory values (stimulus) in bounded time. It receives perceptual and motor schemas from the deliberative level. These schemas are agents running concurrently which use sensory data to calculate the actions to be taken by the actuators. Different motor schemas can calculate actions for the same actuator, and the reactive level must decide in each moment the definitive actions.

The ***S**chemas and **C**ommunications framework*, called SC (Posadas et al., 2002), is a middleware between the reactive and deliberative levels. It provides the hardware and software infrastructure that is required to access sensors and actuators. The deliberative level sends perceptual and motor schemas to the reactive level through the SC framework. The deliberative level accesses the value of the sensors through this middleware and incorporates them to the symbolic internal model of the environment.

The difference in temporal constraints between reactive and deliberative tasks must be considered in the communications framework design. A good real-time performance of communication infrastructure is normally related to predictable computing and a strict frame of communication semantic that is well suited to behavioural activities.

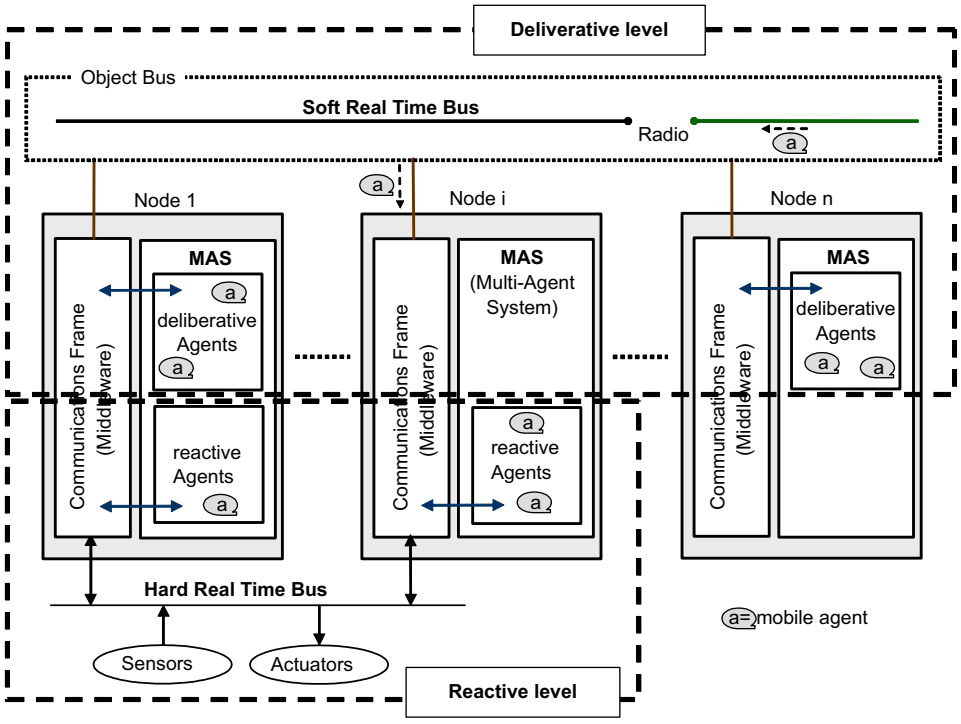

Fig. 2. SC-Agent: multi-level and hybrid architecture that is composed of three distributed parts: a deliberative level, a reactive level and a communications framework.

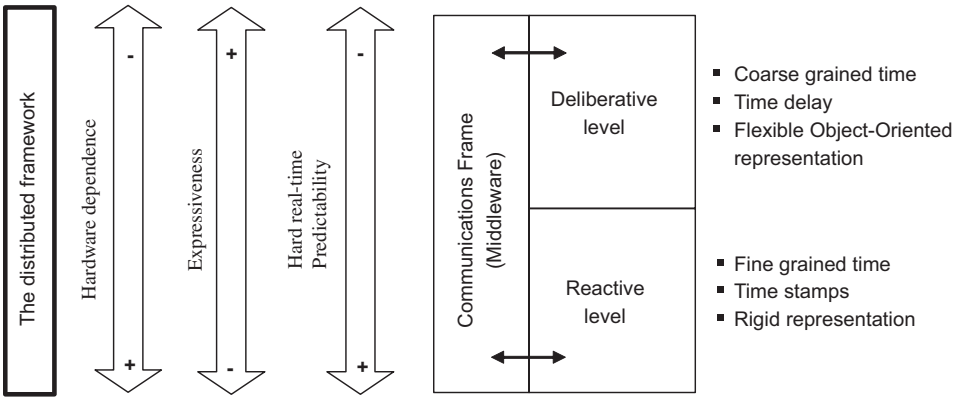

Fig. 3. Characteristics of reactive and deliberative levels.

Other general-purpose, non-real-time communication infrastructures have great semantic flexibility, which is necessary for deliberative computing.

The SC is based on two connection buses: a hard real-time bus which interconnects reactive nodes and a soft real-time bus which interconnects deliberative nodes. Both buses are necessary because real-time restrictions differ (Fig. 3) between reactive and deliberative levels. The reactive level needs a communication bus that guarantees access time and data frame transmission able to respond in bounded time (predictability) when the environment changes. The deliberative level can use a bus without these constraints to communicate more complex data structures such as agents (expressiveness).

Expressiveness can be offered by a domain with flexible object-oriented representation, which usually involves coarse grained time and time delays. However, predictability needs a domain with fine grained time and time stamps, which usually involves rigid representation. The proposed architecture is a solution where the hard real-time bus provides predictability and the soft real-time bus provides expressiveness.

3.1. Guaranteeing hard real-time execution: code delegation

The system must guarantee temporal constraints of hard real-time tasks. This can be difficult because of the following problems:

- *Computing delays*: Processing overload or a lack of resource in the operating system can produce execution delays.

- *Transmission delays*: Some communication channels produce excessive data transmission delays. Communication time cannot be bounded and communication is often impossible because of environmental conditions (i.e.: wireless communication).

The proposed architecture, which is based on mobile agents, reduces temporal constraint problems by separating communication time from processing time. For this reason, it is necessary to take two steps. Firstly, the architecture makes off-line communications (code delegation) in order to move and clone the necessary agents to the robot by using the soft real-time bus—and it then processes the code by using real-time local infrastructure.

The first step is unbounded and the time required depends on environmental conditions. When off-line communications have finished, and the robot has obtained its agents, these are executed locally. This second step can be bounded if agents are executed in physical nodes with hard real-time bus infrastructure. This is because the communication required to get data (on-line communications) through a hard real-time bus can be bounded. As a result, agents will be executed without any external communication that interrupts the processing and real-time restrictions will be guaranteed if the processor achieves the execution periods.

Fig. 4 shows the difference between a robot control that needs to communicate continuously with a central node, and a robot control that uses the proposed architecture. In the first case, the code necessary for the robot to achieve a task is not local and to receive orders the robot needs to communicate continuously with the node where the software is running. In the second case, the necessary code is delegated and then executed locally without external communication delays. In this case, processing time could be bounded.

3.2. Soft real-time execution: time properties of data

Typically, deliberative processes are related to sensory integration, data fusion, and map building. In this case, when temporal and spatial sensory fusion is essential, time properties must be attached to sensor data and control actions.

For example, as stated earlier, sensory data should be available to the planner for the construction of a task-oriented global world model. The construction of this model needs temporal information attached to the distributed objects in order to allow some kind of spatial and temporal data fusion.

So, temporal information about distributed data must be available in all communication infrastructures despite differences in predictability and time granularity. The time property attached to data is in the form of a delay (a time field that accumulates all communication overloads). The main goal of this information is to validate the data in nodes without hard real-time infrastructure. The time field contains the following information:

- t_{delay}: The time elapsed from when data is generated until it arrives at the node.
- $t_{arrival}$: The local time when data arrives at the node.

The SC framework updates the data time field (Fig. 5). When data comes from sensors through the hard real-time bus, the transmission delay can be previously calculated because of the characteristics of real-time buses (Tindell et al., 1994). When this data arrives at a computer, the SC

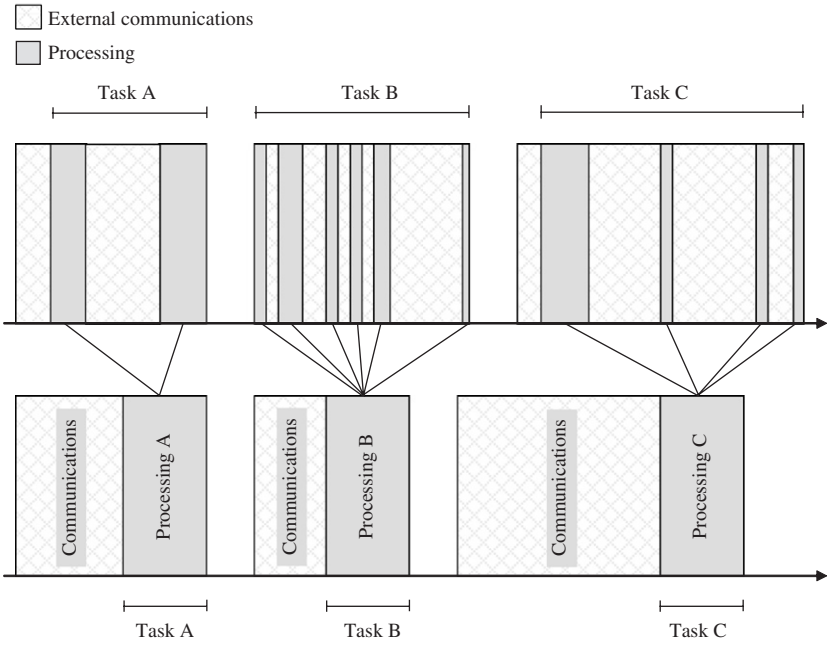

Fig. 4. A robot control that needs to communicate continuously with a central node vs. a robot control where code is delegated and executed locally.

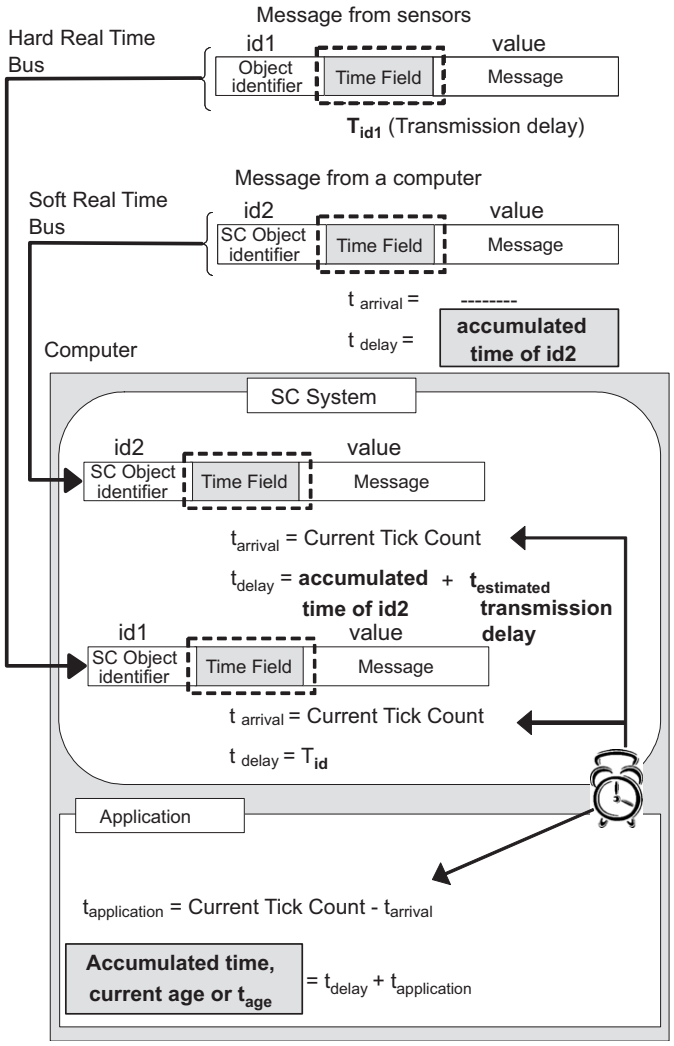

Fig. 5. Calculation of the age of the data. SC updates $t_{arrival}$ with the local tick count when data arrives and t_{delay} by accumulating the transmission delay.

updates its $t_{arrival}$ with the current tick count of the local clock, and its t_{delay} with the transmission delay associated with the data (usually its transmission period). When data comes from another computer through the soft real-time bus, the SC framework has to estimate its transmission delay. When these data arrive, SC updates its $t_{arrival}$ with the current tick count of the local clock, and the SC updates its t_{delay} with the result of adding current t_{delay} (accumulated time before the transmission) and estimated transmission delay.

Each application using data computes its current age and uses it in data integration tasks. Current age of the data (t_{age}) is calculated by adding its t_{delay} to the time that it has remained in the computer ($t_{application}$). This time is calculated by subtracting the current tick count of the local clock and the $t_{arrival}$ of the data.

When data leaves the computer through the SC framework, the time that has remained in the computer is added to the t_{delay}.

3.3. Mobility of components

The software components (mission planning agent, perceptual schemas, and motor schemas) of deliberative and reactive levels are mobile agents which can move among the different nodes. As stated earlier, the main objective of this mobility is to reduce communications overload and communications requirements in the system. Mobility is necessary according to the following criteria:

1. Communications overload and communications requirements depend on the agent's location. Deliberative and reactive levels are distributed. The deliberative level sends various agents to the reactive level which needs different sensory data. Thus, the location of these agents will determine communications requirement and overload.
2. An agent will be able to move to where environmental conditions are better (better execution time, resource availability, lower communication requirements, etc.), this makes it possible to dynamically find an optimal location.
3. Environmental conditions can also change dynamically (operating system resources, hardware availability) and it may be necessary to move the agents to accommodate new environmental conditions.

Mobility introduces greater expressiveness into the system. The proposed architecture provides a new semantic level where agents are specified without having a predetermined location. They could choose their location by using specific measurement indexes. Their location will depend on several factors. An initial factor is based on real-time restriction of their tasks. If the system does not achieve the execution period of the agent, this agent will able to move to another node where it achieves its execution period. A second factor is based on the age of the data that the agent receives as input. Agents need external information and they use the time properties of the data to validate it. When the current age of received data exceeds a maximum value, the agent can move to another node where execution conditions are better. These factors determine the indexes that agents use to determine their movements.

4. Schemas and Communications framework

The SC framework has been developed to access distributed and shared data. This middleware hides the communication details behind a uniform bind-notification interface.

The SC holds an internal representation of the data objects using a distributed blackboard (Nii, 1989). This data structure is continually updated with the changing values of the objects and temporal information is attached. The SC also needs a service running in each node of the system. The SC middleware establishes the required

communications through a soft real-time bus (such as TCP/IP with IIOP messages) to ensure that all the copies of the distributed blackboard are consistent.

Agents communicate between themselves by writing and reading objects into the blackboard (Fig. 6). They must only execute local accesses to contact the system, thus minimizing communication delays. For example, when a perceptual schema needs to obtain the value of a sensor, it only has to read the associated object, which is defined in the local SC service that is running in the same computer as the perceptual schema.

The framework is event driven. Thus, it is possible to associate the code execution (on_message callback, see Section 7.1) with specific events (a change in the value of one object).

Independently of the agent's location and implementation language, agents interact with the SC (Fig. 6) by using a common interface model called frame–sensor–adapter (FSA). A more complete description of FSA is presented in Section 7.1.

The distributed blackboard generated by the SC middleware is extensive to the data in the hard real-time bus (such as CAN bus). This is possible by using a gateway software component that undertakes the communication between SC middleware and the hard real-time bus. The gateway supports communication of the hard real-time bus raw data, as well as the mapped mode that consists of a bi-directional mirroring of hard real-time bus messages and objects in the distributed blackboard (for example, an object linked to infrared messages, etc.). The mapped mode allows processes running in every node of the network to access the hard real-time bus information through the SC framework. The development of gateways to connect the different resources with SC is simple using the FSA model (see Section 7.1).

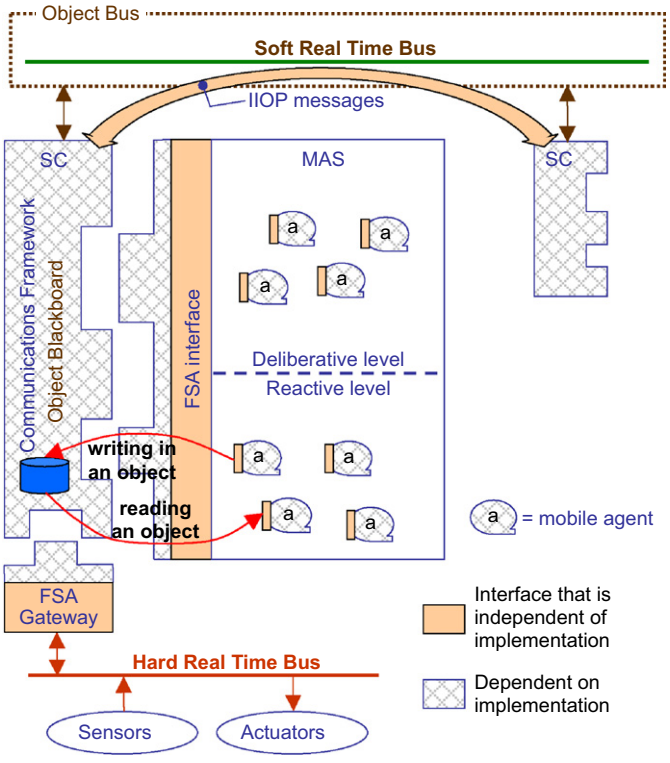

Fig. 6. Interaction between components through SC framework and FSA interface.

5. Reactive level

Reactive and deliberative levels supply a set of behavioural patterns whose composition determines the emergent system behaviour. The reactive level makes up the basic behaviours.

The reactive level is composed of software agents that interact with sensors and actuators through the reading and writing of objects in the distributed SC blackboard. The reactive level is composed of the following components:

1. *Sensors and actuators*: They are interconnected through the hard real-time bus.
2. *Active objects*:
   - *Behavioural patterns*:
     - *Perceptual schemas*: Agents that access the value of the sensors (sensations) and produce sensory information (perceptions).
     - *Motor schemas*: Agents that access perceptual schema data (perceptions) and produce actions to be taken (contributions).
   - *Motivations*: Each motor schema is linked to a motivation process in order to compute the activity of the behaviour. Motivations are agents that access perceptual schema data (perceptions) and weigh motor schema contributions.
   - *Composers*: These are agents that compute definitive actions to be sent to actuators. They use motor schema data (contributions) and associated motivation data (weigh contributions).
3. *Passive objects*: Blackboard objects where agents can read or write data. Most are sensations, perceptions, contributions, motivations and actions.

Agents are concurrent and distributed processes which take local SC data and compute new data independently of what other agents are doing.

Each reactive component (sensor, actuator, perceptual schema, motor schema or motivation) of the architecture is linked to an object in the SC (Fig. 7 and Table 1). The value of this object shows the value of the component. So, there is an object for each reactive component with the exception of composers. Composers are not linked to any object in the SC, but have to provide values to actuator objects.

Each sensor (S_i) of the robot is controlled by a sensory processing module (as an embedded processor) with access to the hard real-time bus. This module periodically

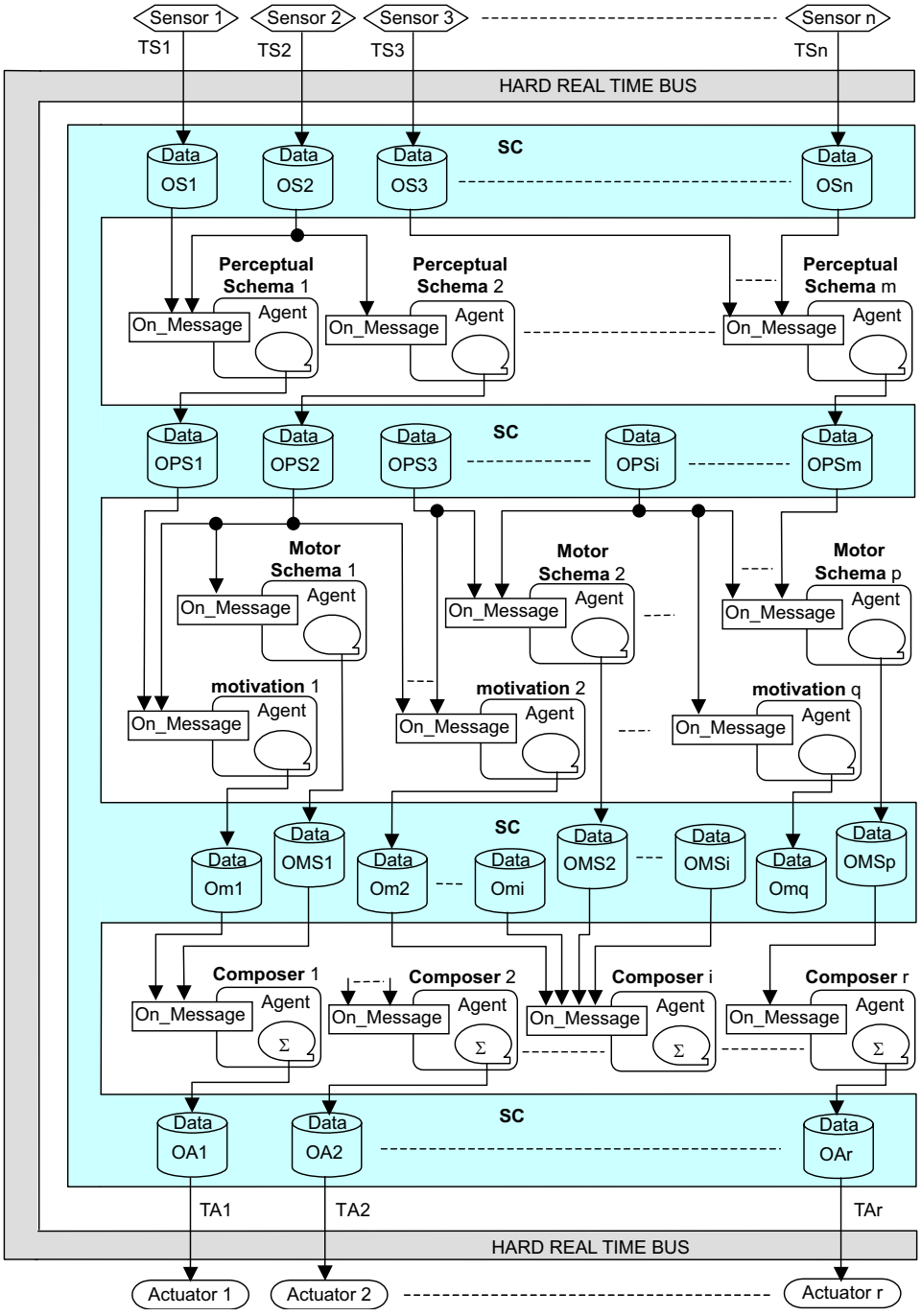

Fig. 7. Reactive level: every sensor, actuator, perceptual schema, motor schema and motivation is linked to an SC object, thereby taking SC data and computing new data.

transmits (with period TS_i) the value of the sensor by sending a message.

Gateway software between SC and the hard real-time bus receives all messages and updates the values of the objects (OS_i) linked with sensors. As a result, all sensor values will be available through the SC framework to all software components at any architectural level.

Each perception schema (PE_i) connects with those SC objects (OS_i) needed to produce a result. For example, a given perception schema could obtain the values of the infrared sensors to determine the presence of an obstacle. The perception schema would then send its result to its associated object (OPE_i) in the SC framework.

Each motor schema (ME_i) obtains the information that it needs from perception schemas by connecting with the corresponding SC objects (OPE_i). With this sensory information, motor schema produces a response or action. For example, when a motor schema that has to avoid obstacles (behavioural pattern) receives information from a perception schema that there is an obstacle ahead, the

Table 1
Nomenclature used in reactive level

Symbol	Description
S_i	Sensor i
TS_i	Transmission period of values from sensor i
OS_i	Object that is associated with sensor i
PS_i	Perceptual schema i
OPS_i	Object that is associated with perceptual schema i
MS_i	Motor Schema i
OMS_i	Object that is associated with motor schema i
m_i	motivation i
Om_i	Object that is associated with motivation i
C_i	Composer i
OA_i	Object that is associated with actuator i
TA_i	Transmission period of values to actuator i
A_i	Actuator i

motor schema will produce the necessary instructions to avoid it. It could order changed velocities in order to go around the obstacle and avoid collision. Each motor schema sends its actions to its linked SC object (OPE_i).

As previously stated, a motor schema is linked to a motivation process (m_i). Motivations are concurrent and distributed software components that weigh motor schema actions. For example, one motor schema may require travel to an objective and another may require a visit to the battery recharge station. The importance of these actions differs depending on the energy level of the robot. Each motivation connects with the SC objects (OPE_i) that it needs to weigh the action of its linked motor schema. The result (a motivation value between 0.0 and 1.0) is sent to the corresponding SC object (Om_i).

Composers (C_i) are the final software component in the reactive level. Composers produce the definitive actions that the robot will perform in each moment. For that, composers send the values to the SC objects (OA_i) that are linked with the actuators (A_i) of the robot.

Each actuator has an associated SC object, the values of this object are the actions that the actuator executes. For example, the motion actuator executes the velocities that are sent to its SC object. In a similar way to sensors, each actuator is controlled by a processing module (as an embedded processor) with access to the hard real-time bus. In this case, the gateway between SC and hard real-time bus periodically sends messages containing the values of the actuator objects (with period TA_i). Each processing module receives its corresponding messages and then executes the actions.

Each actuator is controlled by a composer, although this composer may change. A composer calculates each action for its actuator by using values that motor schemas and associated motivations provide (contributions). A composer is basic or primitive when it only uses the value of a single motor schema to calculate the action that the actuator must execute. A composer is abstract when it uses many values (contributions) from different motor schemas. An abstract composer can be viewed as a set of many different basic behaviour patterns, when each basic behaviour must have a weight in the final action. This weight depends on the mechanism that the abstract composer uses to calculate the definitive action for the actuator. There are two main possibilities (Simó et al., 1997):

- Basic behaviour composition (emergent behaviour). The reactive level makes a weighted addition of all basic behaviours in order to calculate the final action.
- Basic behaviour commutation or sequence. These are situations where all basic behaviours do not have to be considered simultaneously in the calculation of the final action. For example, a robot that has to catch an object—but has to move towards it first. In this situation, behaviours that are necessary to catch the object would have to be considered only after the robot moved towards the object.

In both cases, the values of the motivations are essential and define the activity of each motor schema. When a motor schema has a motivation equal to 0, its contribution will be null in the final action (behaviour commutation can be performed in this manner). In this case, a scheduler process could stop execution of these motor schemas without contribution.

Each composer (basic or abstract) connects with the required SC objects (motor schemas OME_i and motivations Om_i) and sends calculated actions to the SC object (OA_i) of the corresponding actuator.

Reactive agents are provided for at the deliberative level by means of the soft real-time bus. Agents are mobile and can travel through the distributed system. Of course, systems can only guarantee reactive agent execution periods in physical nodes with hard real-time bus infrastructure.

6. Deliberative level

The deliberative level executes planning tasks to reach the assigned objectives of the robot. It divides objectives into subtasks and sends the required software components (perceptual and motor schemas, motivations and composers) to the reactive level.

The deliberative level is also composed of software agents. The components are as follows (see Fig. 8):

- *REA agent*: The component that receives and executes new agents in the system. It is a fundamental component in a MAS where agents are mobile. In SC-Agent architecture, reactive and deliberative agents can travel between system nodes through the SC framework. To achieve this, it is necessary to have an REA agent running in each node of the system. When an agent is going to move to another node or location, its code is written in the blackboard object corresponding to the new location and then the REA agent in the new

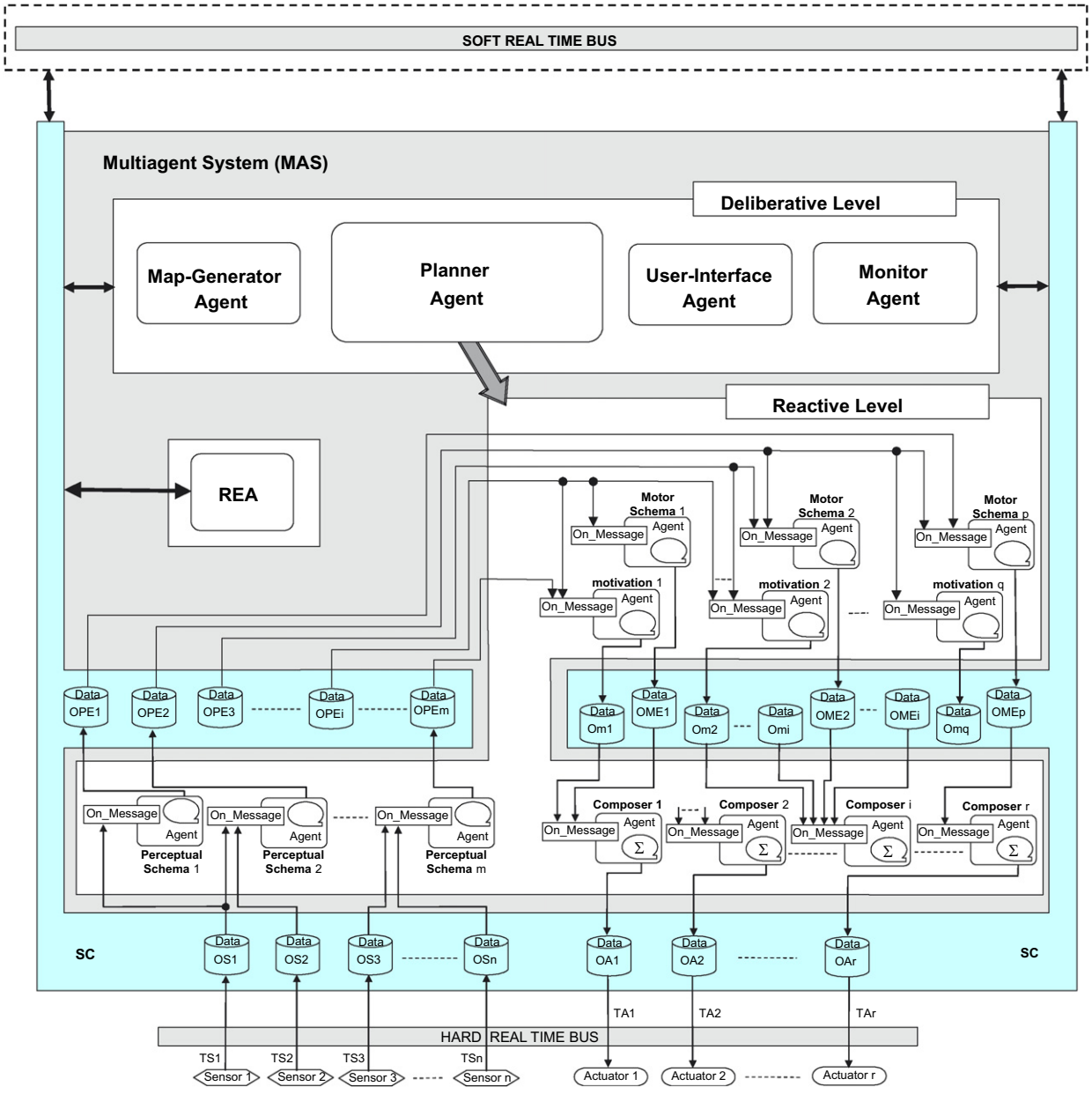

Fig. 8. Deliberative level components.

location receives the code by reading its blackboard object and executing the agent. The SC framework handles the movement transparently through the soft real-time bus.

- *User interface agent*: Interacts with users to receive objectives or tasks that the robot must perform.
- *Mission planning agent*: The main component in the deliberative level. The planning agent plans how to reach robot objectives by using an internal model of the environment. As a result, the planning agent specifies the reactive agents that will have to be executed in the reactive level.
- *Map generator agent*: Constructs the internal model of the environment.
- *Monitor agent*: Supervises system performance and possible errors.

All agents can access the SC distributed blackboard and use all the data to accomplish their functions.

7. SC-Agent in operation

SC-Agent architecture can be applied to control different types of systems, especially mobile robot control systems. This section describes an implementation of the proposed architecture in order to control an explorer robot. The characteristics and functionality of this kind of control can be also applied to other systems such as industrial environments for materials transportation, delivering packages in office buildings, exploration in unknown and hazardous environments, etc. In these systems, code delegation techniques can be useful since robot mission

can be changed remotely by sending new agents without stopping or reinitiating the robot. A set of experiments has been performed using SC-Agent architecture in a real scenario with the following goals:

- To demonstrate mobile robot control by using code delegation techniques.
- To demonstrate the modularity and reusability of software agents in order to implement sensor and motor schemas.
- To demonstrate the use of temporal information attached to data.

First, the common interface FSA (used by agents to access to communications framework) is described. Then, implementation and test results are presented.

7.1. FSA interface

The FSA model has been developed to define a way to access to the SC framework and it extends to any communication resource. It is based on three programming interfaces (Fig. 9): IFrame, ISensor and IAdapter.

- *IAdapter interface*: Defines a set of specific functions that enable access to a communication resource. For each communication resource, an adapter with this interface must be implemented. Agents will use the corresponding *adapter* to connect with the resource. Agents register *sensors* in *adapters* to receive data automatically.
- *ISensor interface*: Defines a set of specific callbacks that enable data to be obtained automatically from a communication resource by using the corresponding *adapter*. A *sensor* is a data receiver and a data holder.
- *IFrame interface*: Defines the framework where *adapters* and *sensors* are located.

The FSA programming model (Fig. 10) requires the implementation of *adapters* based on the IAdapter interface for each communication resource. Agents will use these *adapters* to access communications resources, they will be able to send and receive data by using "*Write*" and "*Read*" functions.

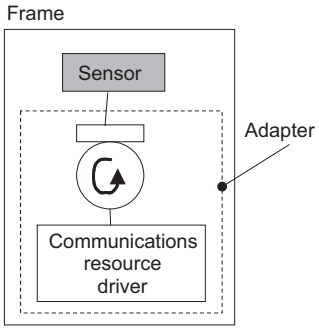

Fig. 9. Frame–sensor–adapter interface.

If agents need to obtain data automatically they must implement *sensors* based on an ISensor interface. Agents must then register the sensors with the *adapters* by using the "*RegisterSensor*" function. When an *adapter* receives data it will automatically execute the "*OnMessage*" callback from registered *sensors*. Agents receive new data through the *sensor* because data are copied in the parameters of "*OnMessage*" callback.

Communications resources are started and cancelled using "*Start*" and "*Stop*" functions (in that moment *adapters* execute automatically "*OnStart*" and "*OnStop*" callbacks from registered sensors). "*GetTimes*" function obtains the current age of received data.

The *adapter* required to access to the SC resource is called SCAdapt. By means of "*RegisterSensor*" function, agents can associate their *sensors* with the objects of the SC blackboard. When the value of one object changes, SCAdapt executes "*OnMessage*" callback and agents receive the new value. Agents change the value of blackboard objects by using "*Write*" function of SCAdapt.

Developing the gateway that connects the "hard real-time bus" with SC was simple using the FSA model (Fig. 11). The "hard real-time bus" resource has its own *adapter*, and gateway component only has to link both adapters (SC and "hard real-time bus") performing specific translations between "hard real-time bus" protocol and SC data.

7.2. Description of the implementation

The architecture has been tested with a real explorer robot (Fig. 12), known as Yet Another Intelligent Robot (YAIR), an autonomous robot with intelligent sensors that measure the environment and its location within it. YAIR was built for the experimental study of reactive systems, sensor fusion, and distributed computing. It has a main computer aboard and it also uses two communication buses:

- A TCP/IP Ethernet bus (soft real-time bus) to communicate via wireless to external nodes.
- A CAN bus (hard real-time bus) to connect different sensors and actuators aboard: a motion controller node, an odometry reckoning node, and an infrared node that supervises a 16 IR detectors ring (it detects objects within a range from 10 to 55 cm.). Sensory information is shared through the bus.

The CAN bus (Tindell et al., 1994) has been chosen for its real-time features. It is a fieldbus initially developed for the automotive industry, yet is currently used in numerous technological areas, especially in mobile robotics—chiefly because of its reliability and versatility. Its medium access mechanism, multi-master capability, and ability to detect transmission errors make it suitable for distributed real-time systems.

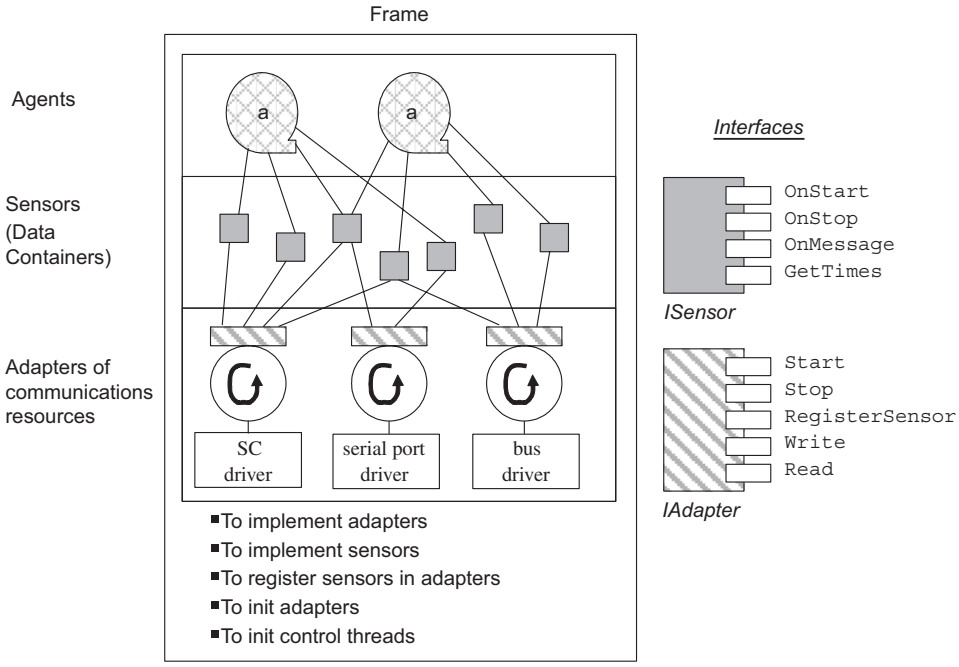

Fig. 10. FSA programming model based on IAdapter and ISensor interfaces.

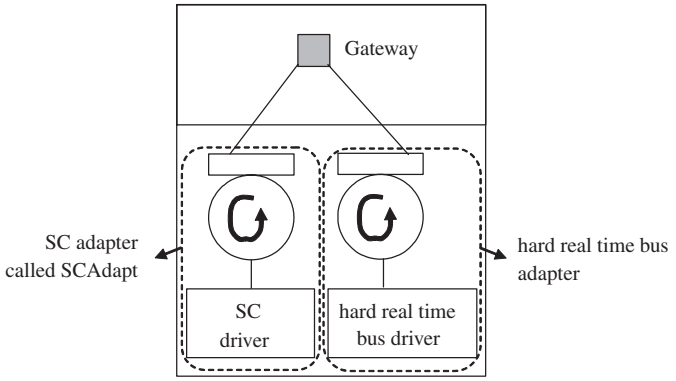

Fig. 11. The gateway model based on FSA links SC and 'hard real-time bus' adapters.

Fig. 12. YAIR robot.

The SC has been implemented by using C++ and needs a service running in each IP node of the system. Communication between SC services is performed through the TCP/IP bus. The distributed blackboard defines an object for every agent to write its results. There is also a blackboard object linked to encoder pulses; another blackboard object linked to infrared values; and, finally, a third blackboard object linked to velocity values.

The SC framework performs specific translations between CAN protocol and SC data through a gateway component called SC-CAN. SC-CAN is executed in the onboard computer, which has access to the CAN bus, and updates periodically the values of the blackboard objects by writing encoder pulses and infrared values which are received from the CAN bus. As a result, information from the CAN bus (encoder pulses and infrared values) will be available through the SC framework to any agent—and even agents running in external nodes without hard real-time bus infrastructure (Fig. 13). Finally, the SC-CAN gateway periodically sends the robot the final

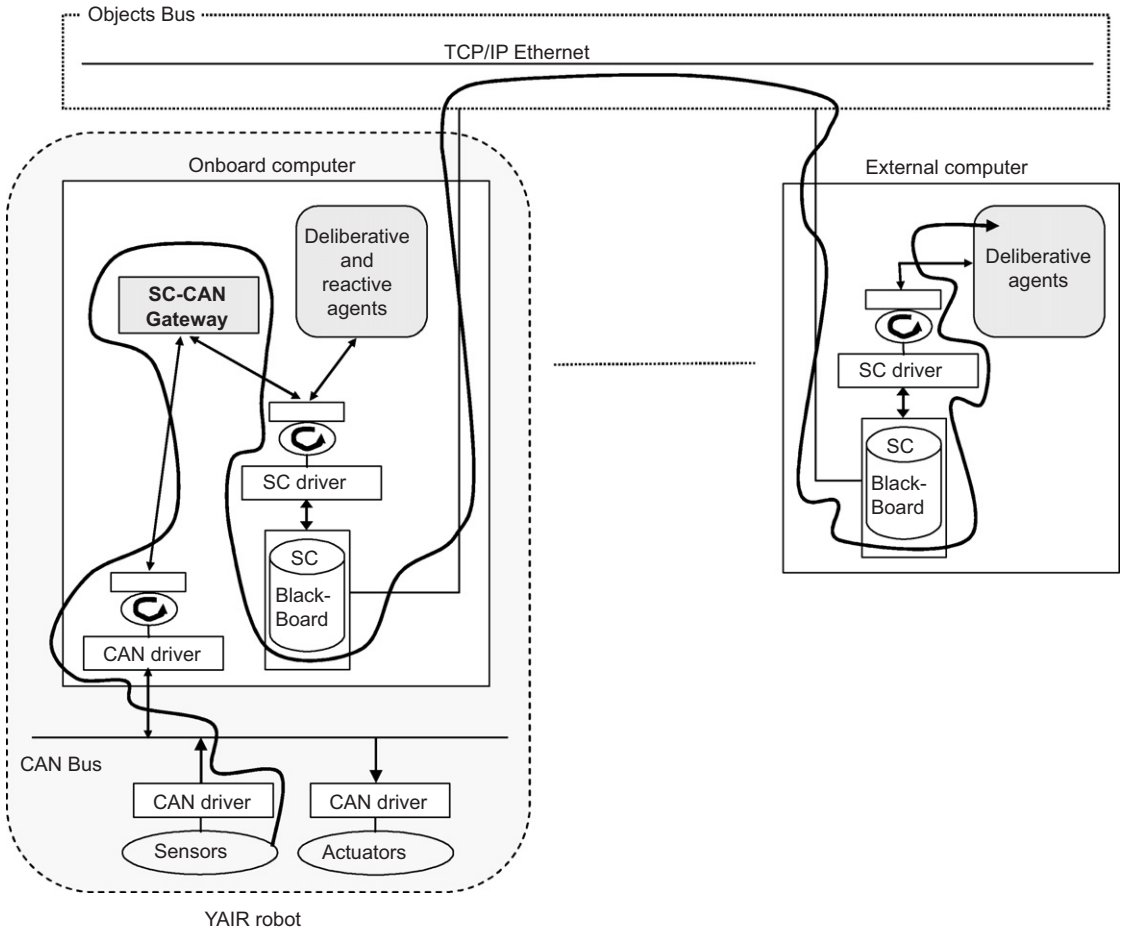

Fig. 13. Agents get sensory data through SC framework. The SC-CAN gateway performs specific translations between CAN protocol and SC data.

velocities which are read from the corresponding blackboard object.

7.3. Developed reactive and deliberative agents

A set of reactive and deliberative mobile agents has been developed using Java language. These agents can run on the onboard computer or in remote nodes—and they access sensor data through the SC framework by using the FSA interface. Experimental tests will use the following two deliberative agents and four reactive agents to control the YAIR robot:

- "*Mission planning and monitor*". This is a *deliberative agent* and central console that orders perceptual and motor schemas to be executed. It shows the results of schemas (covered path and detected obstacles) by drawing the environmental map obtained. This agent needs to read the positions of robot and obstacles—both of which are estimated from perceptual schemas.
- "*Perceptual schema that estimates robot position*". This is a *reactive agent* that reads blackboard objects linked to encoder pulses and calculates robot position.
- "*Perceptual schema that locates obstacles*". This is a *deliberative agent* that reads blackboard objects linked to infrared sensor values and robot positions and calculates the position of the obstacles by fusing sensory data. It uses infrared values to detect obstacles and makes couplings between detected obstacles and robot positions by using time attached to data (the age of the data). Interpolation techniques are used to calculate precise robot position.
- "*Motor schema that avoids obstacles*". This is a *reactive agent* that reads blackboard objects linked to infrared sensor values and calculates velocity values (contribution) for left and right wheel motors to avoid possible obstacles by applying the Braitemberg algorithm.
- "*Motor schema that explores environment*". It is a *reactive agent* that reads blackboard objects linked to robot positions and calculates velocity values (contribution) for left and right wheel motors to explore the environment.
- "*Composer*". This is a *reactive agent* that uses a weighing process to determine robot movements. The composing process consists of a weighted addition of motor schema values (contributions) in order to obtain definitive velocity values.

A *REA deliberative agent* that receives and executes the code of mobile agents has also been developed. This agent also needs a service running into each IP node of the system. An agent can move to another node by writing its code on the SC blackboard object that is linked with the REA agent that is running on the remote node. Firstly, the REA agent receives the code by reading the associated SC blackboard object—and it then executes the mobile agent code in its new location.

During the tests the aim is to control the robot moving in an unknown environment using behavioural pattern composition. As stated before, the final calculation of the actions is affected by motivation values. This implementation follows the easiest model of emergent behaviour, that is, it considers that motivation values are constant (with value 1). So, the emergent behaviour of the YAIR robot is the result of motor schema composition by means of the single addition of the values of each motor schema. These motor schemas must be compatible in order to avoid conflicting compositions.

Agent reusability is desirable in robotics. Using the same agents to control similar robots is possible with the SC-Agent architecture. In fact, the above-described agents were developed to control a Khepera robot (Posadas et al., 2004) in an initial approach. A Khepera robot does not have a hard real-time bus but it can be useful to test preliminary deliberative level aspects. It sends sensory values through a serial bus; and thus a gateway to translate information between the serial bus and SC framework was developed. The above-described agents were executed in distributed nodes to control the robot and demonstrate data access through the SC framework. These agents were easily reused on the YAIR robot by just modifying the configuration archive that describes sensor characteristics (number of sensors, characteristic equation, distance, etc.) and robot characteristics (radius, distance between wheels, encoder pulses, etc.).

An important aspect to be highlighted is that reactive components can be sent (code delegation) to the robot from external nodes. So, deliberative agents can be executed from an external node and so can send new reactive agents to the robot to be executed locally.

7.4. Results obtained

The parameters used to compare the different approaches tested to control a mobile robot were: covered path, detected obstacles, elapsed time and robot velocities. Parameters that could be modified in each test were: agent execution periods and agent location (an agent can be executed locally in the robot's onboard computer—or it can be executed remotely in an external node).

Agent execution periods determine the maximum speed of the robot. So, the greater the execution frequency—the greater the robot speed. However, minimal periods are established by hardware restrictions, such as the frequency the nodes can send encoder values or infrared values. The maximum speed of the robot is bounded by the maximum distance that an obstacle can be detected by sensors. As a result, agent execution periods and maximum speed must be adjusted to the hardware of the robot and then used in all tests. An emergency stop is performed when the system cannot achieve execution periods and the robot then remains motionless.

Agent location determines elapsed time and used velocities to run a mission. The number of emergency stops made can be reduced by adequately distributing software agents.

SC-Agent architecture allows differing methods of distributing agents at runtime. The aim of the tests is to compare results when all of agents are executed locally (test 1); when all agents are executed remotely (test 2); or when agents are distributed between local and remote nodes depending on their reactive or deliberative tasks (test 3). Table 2 shows used agent periods and locations (local or remote) in each test.

Table 2
Agent periods and locations (local or remote) in each test

Reactive and deliberative agents		Period (ms)	Location		
			Test 1 (all local)	Test 2 (all remote)	Test 3 (distributing)
SC-CAN gateway	To receive and send infrared values from CAN to SC	60	Local execution in computer onboard with access to CAN bus		
	To receive and send encoder pulses from CAN to SC	50			
	To receive and send velocities from SC to CAN	400			
Perceptual schema that estimates robot position		50	Local	Remote	Local
Motor schema that avoids obstacles		100	Local	Remote	Local
Composer		200	Local	Remote	Local
Motor schema that explores environment		100	Local	Remote	Local
Perceptual schema that locates obstacles		–	Local	Remote	Remote
Mission planning and monitor		–	Local	Remote	Remote

In all tests, the mission of the robot was to explore the environment and to locate obstacles. Fig. 14 shows an information flow diagram of the proposed control solution applied.

The robot is initially positioned at the centre of the environment and control agents start to work without having any preliminary information about the environment. The system does not know the path the robot will

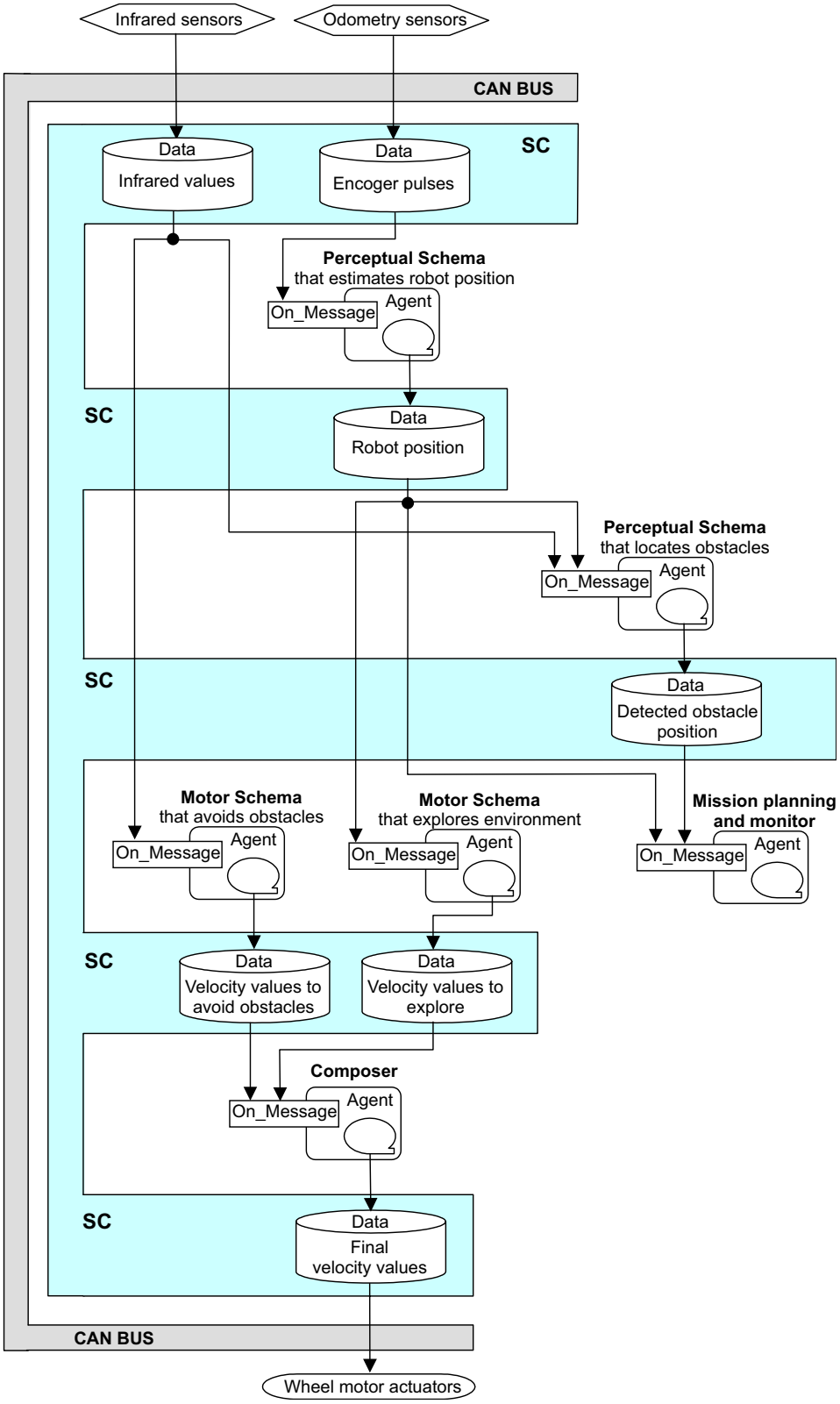

Fig. 14. Information flowchart.

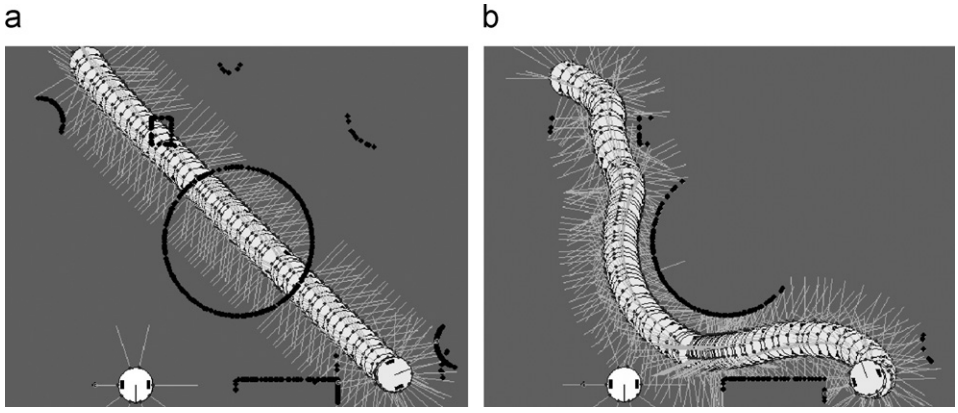

Fig. 15. The differences between using or not the composing process.

follow. It will depend on the composing process. According to the robot position and following a simple exploration algorithm (up and down and left to the right) the explorer agent calculates velocity values to cover the area. In parallel and independently of what this agent is doing, the agent that avoids obstacles calculates other velocity values according to infrared sensor values. In the last step, the composer agent obtains definitive velocity values calculating throughout a weighted addition of the above mention values. As a result, the robot moves autonomously by the emerging of an intelligent behaviour of the interaction of two simple behaviours—robot moves to explore avoiding possible obstacles. An important aspect to be highlighted is that this emergent behaviour is not planned. Therefore, the covered path, obtained step by step as a result of the composing process, is not fixed. It can change in every test. Fig. 15 shows composing process results by using a simulator (Posadas et al., 2004). If the robot had not the agent to avoid obstacles (it only counts on the explorer agent), the robot would not move properly and would crash with obstacles (Fig. 15a). Both agents and the composing process are necessary for the robot to move avoiding obstacles (Fig. 15b) behaving then in an intelligent manner.

In all tests, the real environment and obstacles used (Fig. 16) are fixed in order to compare elapsed time results. The aim is to establish in which test it takes the robot less time to finish its mission. However, control agents would work in a dynamic environment with mobile obstacles in the same way described in the above paragraph. No changes would be necessary since sensors would be able to detect mobile obstacles the same way they detect motionless obstacles. Therefore, the main difference in workspace environments for each test is the agent location. However, despite of the fact that the environment and obstacles are fixed, the covered path in every test shows little differences due to dynamic aspects such as the sensor response, slides, frictions, etc., that leads to variations in the emergent behaviour.

Test results are presented in Fig. 17a–c. Each figure shows, on the left, the robot environment with the robot path covered and the detected obstacles. On the right, a graph with the elapsed time and velocity values for left and right wheel motors is shown.

Fig. 16. Real environment explored in all tests.

In test results, a similar path covered and same obstacles detected can be seen. As stated out before, this is due to the fact that the obstacles and the explorer algorithm are the same in every test. Differences mainly appear in the elapsed time that is needed to explore the same environment.

In test 1 (Fig. 17a), all agents were executed locally. The problem with locally executing all agents is that the system could overload because of the amount of deliberative processing (in such a case, the system would not achieve the execution periods of reactive agents and would become unstable). In our case, when the system becomes unstable an emergency stop is performed. If the processor is overloaded, several time intervals where the robot is stopped (velocities equals zero) will appear in the graph. SC-Agent architecture enables the robot to finish its mission but it takes more time because robot only moves when the onboard computer can achieve reactive execution periods.

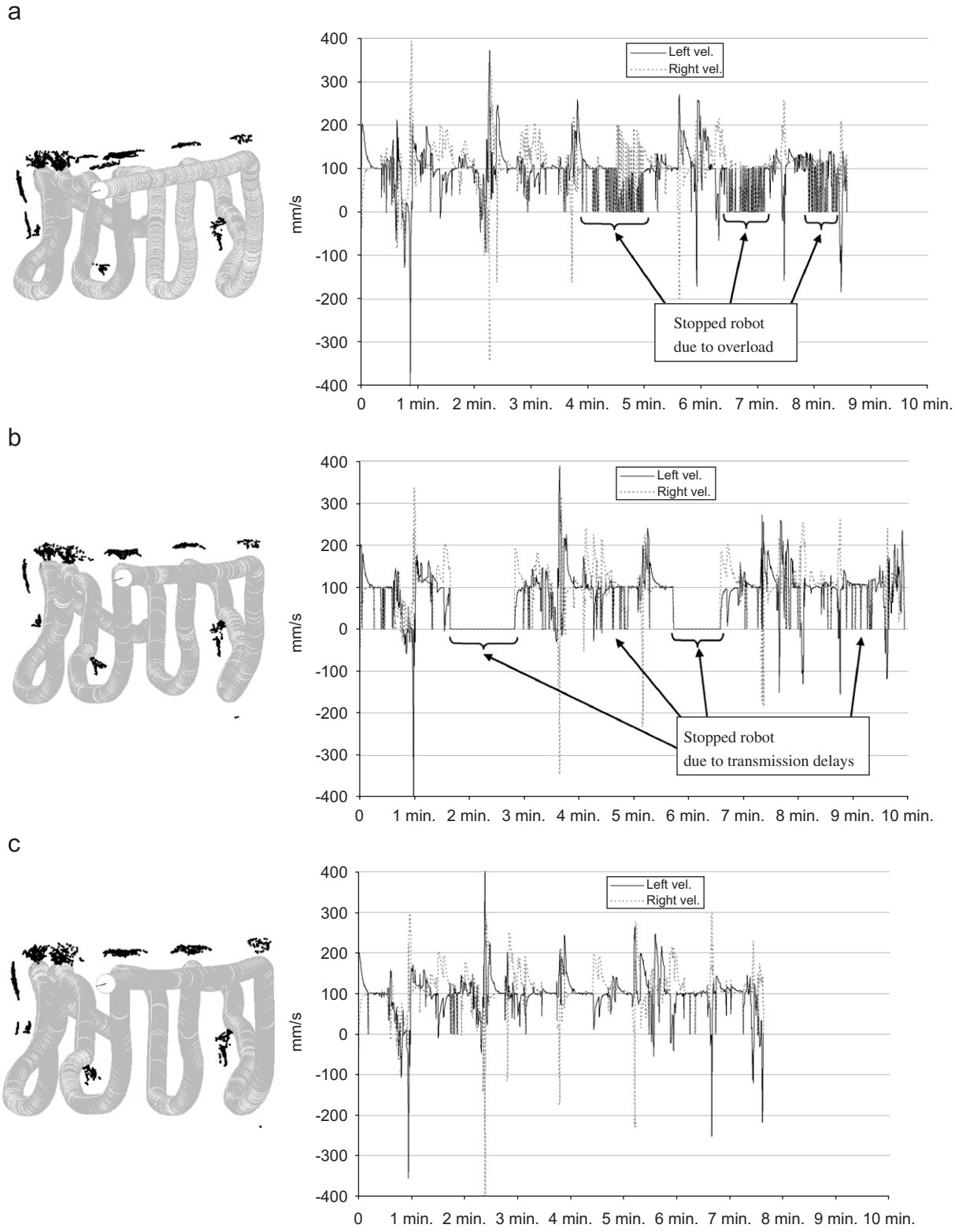

Fig. 17. Results in tests 1–3.

In test 2 (Fig. 17b), all agents are executed remotely. This type of approach works fine when there is little delay in communication. However, when transmission delay is introduced, the system becomes unstable and emergency stops are performed. In graph two time intervals where the robot is stopped (velocities equals zero) can be seen corresponding to this situation. In the test, this situation delays the robot mission for some minutes.

The distribution of the agents depending on their real-time restrictions could be a solution for detected problems

in tests 1 and 2. The aim is to locally execute the reactive agents and remotely execute the deliberative agents. First, reactive agents from remote nodes are sent to the onboard computer. The reactive agents are then executed locally without any delay in communication. The local processor will not overload because the deliberative agents are executed in remote nodes. The results of the test conducted using this solution can be seen in Fig. 17c. It shows that robot missions take less time and there is no time interval when the robot is stopped.

These results illustrate the advantages of controlling mobile robots by using code delegation techniques. A suitable communication framework which enables the flow of processes between nodes and also enables distributed data to be obtained from any location, is also required to distribute agents. Besides, temporal information attached to data is necessary to enable agents to fuse sensory data, in the same way as the described "*Perceptual schema that locates obstacles*". The effectiveness of SC-Agent architecture and the SC framework, based on these requirements, has been demonstrated in the above-described test 3.

8. Conclusions

Communications are one of the most difficult parts of building robot architecture and it is an area that is often ignored. This paper is an introduction to the area and gives some guidelines for architecture implementers.

The main contribution of this work is a modular and general hybrid architecture that can be applied to control different types of systems, especially mobile robot control systems. A middleware or communications framework is the backbone of the architecture—allowing interaction between all agent components. Real-time restrictions of reactive and deliberative levels are considered in the design of the architecture by using different connection buses to guarantee temporal constraints. The proposed architecture based on SC middleware is easily adaptable to any number of sensors and actuators and can be implemented quickly. Behavioural patterns (perceptual and motor schemas), motivations and composers are mobile agent components that can be dynamically added to the system. Architectural implementation is based on implementation and instantiation of necessary agents. The proposed architecture provides a new semantic level where processes are specified without a predetermined location. They can move, depending on their deliberative and reactive tasks, to where environmental conditions are better and there is less communication overload.

The following advantages can be seen regarding code delegation techniques: each agent can be implemented independently and new agents can be quickly and dynamically added without stopping the system. Robot control can be made by sending the necessary agents; robot mission can be changed easily by sending new agents without reinitiating the robot; and networking latency problems can be reduced as there is no communication delay when agents are delegated because the processing is local.

The results of the tests described illustrate the advantages of controlling mobile robots by using code delegation techniques. SC-Agent architecture allows reactive agents to be executed locally without any delay in communication and the local processor will not overload because deliberative agents can move to be executed in remote nodes.

The deliberative agents are commonly expressed using rule-based languages. To add explicitly classical rule-based agents in deliberative levels we have explored the development of a Domain Specific Language for event processing. The agent's specification is a XML file that can be executed and managed by specific engines. Currently we have adapted engines based on CLIPS and DROOLS. The integration of these and other Artificial Intelligence engines into the proposed architecture is straightforward and part of the future work.

The autonomous systems and mobile robots are currently applied in many industrial areas such as intelligent transportation, indoor offices environments delivering and surveillance, exploration and physical interaction in unknown and hazardous environments among many others. The proposed architecture is intended to the development of intelligent hybrid and autonomous systems and fits well into the development of cyber–physical systems and therefore into the above-mentioned applications.

References

Alami, R., Chatila, R., Fleury, S., Ghallab, M., Ingrand, F., 1998. An architecture for autonomy. International Journal of Robotics Research, special issue on "Integrated Architectures for Robot Control and Programing".

Albus, J.S., 2002. 4D/RCS a reference model architecture for intelligent unmanned ground vehicles. In: Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defence Sensing, Simulation and Controls, Orlando, FL.

Arbib, M., 1986. Schema Theory, The Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge, MA.

Arkin, R., 1990. Integrating behavioural, perceptual and world knowledge in reactive navigation. Robots and Autonomous Systems 6, 105–122.

Arkin, R., 1998. Behaviour-Based Robotics. MIT Press, Cambridge.

Bonasso, R.P., Firby, J., Gat, E., Kortenkamp, D., Miller, D., Slack, M., 1997. A proven three-tiered architecture for programming autonomous robots. Journal of Experimental and Theoretical Artificial Intelligence 9 (2).

Brennan, R.W., Fletcher, M., Norrie, D.H., 2002. An agent-based approach to reconfiguration of real-time distributed control systems. IEEE Transactions on Robotics and Automation 18 (4) (August).

Brooks, R.A., 1986. A robust layered control architecture for a mobile robot. IEEE Journal of Robotics and Automation 2 (1), 14–23.

Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A., 2005. Towards component-based robotics. In: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 163–168.

Brooks, A., Kaupp, T., Makarenko, A., Williams, S.B., Oreback, A., 2007. Orca: a component model and repository. In: Brugali, D. (Ed.), Software Engineering for Experimental Robotics. Springer, Berlin, pp. 231–251.

Brugali, D., Fayad, M.E., 2002. Distributed computing in robotics and automation. IEEE Transactions on Robotics and Automation 18 (4) August.

Colon, E., Sahli, H., Baudoin, Y., 2006. CoRoBa, A multi mobile robot control and simulation framework. International Journal of Advanced Robotic Systems 3 (1), 73–78.

Gowdy, J., 2000. A qualitative comparison of interprocess communications toolkits for robotics. Technical Report CMU-RI-TR-00-16, Robotics Institute, Carnegie Mellon University, June.

Hassan, H., Simó, J., Crespo, A., 2001. Flexible real-time mobile robotic architecture based on behavioural models. Engineering Applications of Artificial Intelligence 14, 685–702.

Janglova, D., Uher, L., Vagner, S., 1996. Control of mobile robot with reflexive behavior. In: Proceedings of Fifth International Workshop on Robotics in Alpe-Adria-Danube Region RAAD'96, Budapest, Hungary, pp. 603–607.

Janglova, D., Vagner, S., 1999. Intelligent mobile robot for indoor environment. In: Proceedings of the conference Automation'99, Warsaw, Poland, pp. 233–238.

Jennings, N.R., Wooldridge, M.J., 1998. Agent Technology. Springer, Berlin.

Konolige, K., Myers, K., 1998. The Saphira architecture for autonomous mobile robots. In: Kortenkamp, D., Bonasson, R., Murphy, R. (Eds.), Artificial Intelligence and Mobile Robots. MIT Press, Cambridge, MA.

Kopetz, H., Nossal, R., 1997. Temporal firewalls in large distributed real-time systems. In: Proceedings of Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems. IEEE Press, New York.

Lange, D.B., Oshima, M., 1999. Seven good reasons for mobile agents. Communications of the Association of Computing Machinery 42 (3).

Murphy, R.R., 2000. Introduction to AI Robotics. MIT Press, Cambridge.

Nii, H.P., 1989. Introduction. In: Jagannathan, V., Rajendra, D., Lawrence, S.B. (Eds.), Blackboard Architectures and Applications. (Perspectives in Artificial Intelligence), vol. 3. Academic Press, Boston, pp. 19–29.

Nilsson, N.J., 1980. Principles of Artificial Intelligence. Morgan Kaufmann Ed, Los Altos, CA.

Orebäck, A., Christensen, H.I., 2003. Evaluation of architectures for mobile robotics. Autonomous Robots 14, 33–49.

Posadas, J.L., Perez, P., Simo, J.E., Benet, G., Blanes, F., 2002. Communications structure for sensory data in mobile robots. Engineering Applications of Artificial Intelligence 15, 341–350.

Posadas, J.L., Simó, J., Blanes, F., Benet, G., Poza, J.L., 2004. An architecture to control mobile robots by means of code delegation and multi-agent systems. In: Fifth IFAC/EURON Symposium on Intelligent Autonomous Vehicles (IAV), Portugal.

Schlegel, C., 2006. Communication patterns as key towards component-based robotics. International Journal of Advanced Robotic Systems 3 (1), 49–54.

Simó, J., Crespo, A., Blanes, J.F., 1997. Behaviour selection in the YAIR architecture. In: IFAC Conference on Algorithms and Architectures for Real-time Control, Portugal.

Tindell, K.W., Hansson, H., Wellings, A.J., 1994. Analysing real-time communications: controller area network (CAN). In: Jahanian, F., Ramaritham, K. (Eds.), Proceedings of IEEE RealTime Systems Symposium RTSS'94 Proceedings of IEEE RealTime Systems Symposium RTSS'94. IEEE Computer Society Press, Silver Spring, MD, pp. 259–263.