A real-time distributed software infrastructure for cooperating mobile autonomous robots

Frederico Santos [1,3]
[1] DEE - ISEC
I. P. Coimbra, Portugal

Luís Almeida [2,3]
[2] DEEC - FEUP
Univ. Porto, Portugal

Paulo Pedreiras [3], Luís Seabra Lopes [3]
[3] IEETA - DETI
Univ. Aveiro, Portugal

Abstract— **Cooperating mobile autonomous robots have been generating a growing interest in fields such as rescue, demining and security. These applications require a real time middleware and wireless communication protocol that can effecient and timely support the fusion of the distributed perception and the development of coordinated behaviors. This paper proposes an affordable middleware, based on low-cost and open-source COTS technologies, which relies on a real-time database partially replicated in all team members, containing both local and remote state variables, in a distributed shared memory style. This provides seamless access to the complete team state, with fast non-blocking local operations. The remote data is updated autonomously in the background by a WiFi-based wireless communication protocol, at an adequate refresh rate. The software infrastruture is complemented with a task manager that provides scheduling and synchronization services to the application processes on top of the Linux operating system. Such infrastructure has been successfully used for four years in one RoboCup middle-size soccer team, and it has proved to be dependable in the presence of uncontrolled spurious traffic in the communication channel, using an adaptive technique to synchronizating the robots in the team and reconfiguring the communications dynamically and automatically according to the number of currently active team members.**

I. INTRODUCTION

Coordinating several autonomous mobile robotic agents in order to achieve a common goal has been an active topic of research for more than a decade [1][2]. This problem can be found in many robotic applications, either for military or civil purposes, such as search and rescue in catastrophic situations, demining or maneuvers in contaminated areas. The technical problem of building an infrastructure to support the perception integration for a team of robots and subsequent coordinated action is common to the above applications. One initiative that was created to promote research in this field is RoboCup [3] Middle Size League where several autonomous robots have to play soccer together as a team, to win a match against another team of autonomous robots.

Currently, the requirements posed on such teams of autonomous robotic agents have evolved in two directions. On one hand, robots must move faster and with accurate trajectories to close the gap with the dynamics of the processes they interact with, e.g., a ball can move very fast. On the other hand, robots must interact more in order to develop coordinated actions more efficiently, e.g., only the robot closer to the ball should try to get it while other robots should move to appropriate positions. The former requirement demands for tight closed-loop motion control while the latter demands for an appropriate communication system that allows building a global knowledge base to support cooperation.

Different middleware layers have been developed to help the task of programming teams of autonomous agents, providing logical abstractions to support cooperation [2]. Unfortunately, the actual use of communication and synchronization by the specific middleware layer may impose different delays and, in the end, may cause the middleware to fail supporting the requirements referred above.

Therefore, to support such requirements efficiently, a specific software infrastructure was developed for the CAMBADA (Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture) middle-size robotic soccer team of the University of Aveiro, Portugal, which is composed by three main components: a middleware based on a Real-Time Database (RTDB); a wireless communication protocol based on WiFi and implementing a Reconfigurable and Adaptive TDMA (RA-TDMA), and; a process manager (PMan) to provide enhanced synchronization (real-time) services to the general purpose Linux Operating System.

The former was initially proposed in [4] and follows a distributed shared memory model that provides each agent with a local copy of the state variables of the other agents. These local copies are updated transparently and autonomously in the background by the communication protocol, which has several adaptive and reconfigurable properties that reduce intra team collisions and grant it higher robustness with respect to external interferences [5]. Particularly, the communication protocol is fully distributed, using minimal configuration parameters, thus being very easy to deploy. Finally, the PMan [6] provides services for the Linux OS that are typical of Real-Time Operating Systems (RTOS) namely support for automatic activation of recurrent tasks with good precision, phase control and precedence constrains.

Despite having been developed for a robotic soccer team, this middleware is equally suited to situations in which a team of robots must interact tightly to achieve a comon goal, from search and rescue, to surveillance and pursuit. We will also show that it is an affordable and dependable middleware in the sense that it is based on COTS hardware and open source software and it is robust to spurious transmissions in the channel using an adaptive phase rotation-based mechanism to keep the team robots synchronized and reconfiguring the communications cycle dynamically according to the number of currently active robots.

This paper presents the three middleware components in

an integrated way, with a focus on the RTDB middleware, its current implementation, which has evolved considerably since its initial version in [4], and its relationship with other similar infrastructures. This relationship is discussed in Section II, with the RTDB being presented in Section III. The communication protocol is briefly described in Section IV and the PMan in Section V. Section VI addresses timing issues, namely the age of the data within the RTDB. This Section also includes an analysis of the communication delays. Finally, Section VII concludes the paper.

II. RELATED WORK

Similarly to other RoboCup middle-size league teams [7][8], our team software architecture emphasizes cooperative sensing as a key capability to support the behavioral and decision-making processes in the robotic players. A common technique to achieve cooperative sensing is by means of a *blackboard* [9], which is a database where each agent publishes the information that is generated internally and that may be fetched by others.

Blackboard-based middlewares are frequently built in-house, resulting in many flavors. Nevertheless, one typical approach is based on the client-server (CS) cooperation model and thus, when a robot needs a datum, it has to communicate with the server holding the blackboard. This introduces an undesirable delay at the time the datum is required, which depends on the current network availability and utilization level as well as the interval between when the datum was written and when it is requested. This model does not necessarily consider synchronization between the data producers that write in the blackboard and the clients that access it, which can increase the age of the data.

These CS approaches are frequently built on top of the Common Object Request Broker Architecture (CORBA) distribution middleware, which provides independence with respect to the data actual physical location. Examples of such include Miro [10][11], RT-Middleware [12] and RSCA [13]. These and other middlewares have been compared in [2] according to a list of relevant objectives, namely simplification of the development process, reusability, integration, flexibility, self-discovery, self-configuration and QoS support. Such work also concludes that it is difficult for a unique middleware to fulfil all the enumerated objectives and that the provided characteristics are a balance between the application domains and the robotic systems. In general all middlewares accelerate the development process providing distribution abstractions that simplify the development of coordinated behaviours. The other objectives referred are also commonly met by the analysed middlewares, with the exception of QoS support, which is only provided by RSCA that makes use of RT-CORBA approach for the communication.

A different cooperation model is the publisher-subscriber (PS) [14] in which the entities associated with a given datum register in a group. The producers of such datum are the publishers and those needing the datum are the subscribers. The middleware layer takes care of the data dissemination, sending the data generated by the publishers to the respective subscribers in a multicast fashion. PS middlewares are also well suited to implement blackboards. In this case, publishers place data directly in the blackboard and the middleware automatically sends it to the registered subscribers with a small latency, thus being better suited to meet real-time requirements than client-server models. A well known PS middleware is the Data Distribution Service (DDS) [15]. A key aspect of this middleware is the use of QoS parameters to configure the system and establish contracts between publishers and subscribers specifying exactly how information should flow between the nodes. QoS contracts provide the performance predictability and resource control required by real-time systems while preserving the modularity, scalability and robustness inherent to the anonymous PS model. OpenRDK [16] is an open source middleware, that aims at extending the Quality of Service with the inclusion of additional features inspired in the DDS specification.

Another model that is rather similar to the PS one is the producer-consumer (PC) [17]. The main difference is that it is based on broadcast, thus involving all nodes at a time. Whenever a producer generates data, it is made available to all potential consumers. Those that are the actual consumers of a given datum identify it and retrieve the datum from the network interface. Both PS and PC middlewares are well suited to implement the distributed shared memory model [18] in which each node has local access to all the process state variables that it requires. Those variables that are remote have a local image that is updated automatically by an autonomous communication system.

In our target application, i.e., the coordination of a team of mobile robots, the network must necessarily be wireless. When comparing with wired networks, wireless ones present higher and asymmetric bit error rates, leading to more packet drops and to connectivity and network availability losses. It is thus important that the middleware handles these limitations. The unicast nature of CS middlewares allows them to use acknowledges and automatic retransmissions, which tend to improve the reliability of the communications. However, retransmissions can also cause data to become old beyond its validity, thus becoming useless. On the other hand, PS or PC middlewares typically use one to many unacknowledged communications, thus without retransmissions. Packet drops are ignored and compensated with subsequent transmissions that are normally periodic. The absence of retranmissions keeps the network load generated by the team always at the same level, which has a stabilizing effect in the network. Conversely, retransmissions imply an increase in the network load, possibly leading to thrashing. In this sense, PS and PC middlewares can be considered more robust, thus more adequate to wireless communication than CS ones. Nevertheless, the recovery from a packet loss is typically faster with retransmissions as long as the network load is not too high, which might favor CS approaches.

Common robotics middleware does not take care of the wireless communication problems or even tries to share the medium with traffic from other sources, but focus mostly on the interaction between modules, using CORBA imple-

mentations that are heavy-weight and introduce additional complexity in the network. SPICA [19] is an example of a middleware that tries to solve the problem using a simple, lean and fast communication infrastructure, but for usage in ad-hoc communications.

In this paper we propose a PC middleware that shares the features of that class but goes a step further presenting a few novel features that are particularly adapted to the coordination of teams of mobile robots. Particularly, it implements the distributed shared memory model giving each node seamless access to remote variables as if they were local, abstracting away both distribution and communication, it includes a specific communications protocol based on a reconfigurable and adaptive TDMA approach that minimizes the collisions among team members and further contributes to the network stability in a shared medium with other sources of traffic. It also includes a task manager that provides enhanced synchronization services to tasks executing on a general purpose operating system within each node. Overall, the proposed middleware is affordable since it is based on COTS hardware technologies and open source software, it is dependable in the sense that it is robust to transmission errors and to spurious transmissions and it meets most of the objectives referred in [2] namely, simplification of the development process, reusability, integration and QoS.

III. THE RTDB

Similarly to the concept presented in [20], we developed a replicated blackboard called Real-Time Database (RTDB), which holds the state data of each agent together with local images of the state data shared by other team members. A specialized communication system triggers the required transactions in the background at an adequate rate to guarantee the refresh of those local images.

In the robotic soccer case, the information within the RTDB holds the absolute positions and postures of all team members, as well as the position of the ball, among other less relevant data. This approach allows a robot to easily use the other robots sensing capabilities to complement its own. For example, if a robot temporarily loses track of the ball it might use the position of the ball as detected by another robot. This is done without explicit use of communication, abstracting away the data distribution itself.

A. RTDB Implementation

The RTDB is fully implemented in `ANSI C` over several blocks of shared memory. One of the blocks is a private area for local information, only, i.e., which is not to be disseminated to the other robots; and the other blocks (one corresponding to each team member) are the shared area with global information (see Fig. 1). One of the shared blocks is written by the agent itself (read-write), whose data is sent to the others and could also be used for interprocess communication, while the remaining blocks (read-only) are used to store the information received from the other agents.

The allocation of shared memory is carried out by means of a specific function call, `DB_init()`, called once by every process that needs access to the RTDB. The memory allocation is executed by the first process to use such call, only. Subsequent calls just return the shared memory block handler and increment a process count. Conversely, the memory space used by the RTDB is freed using the function call `DB_free()` that decreases the process count and, when zero, releases the shared memory block.

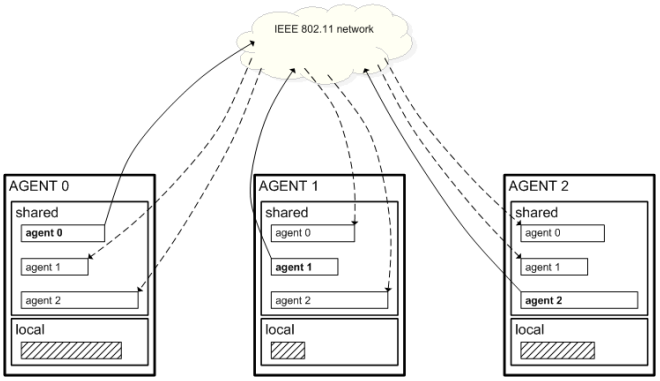

Fig. 1. Each agent transmits periodically its subset of state data that might be required by other agents

The RTDB is accessed concurrently by processes that capture and process images and implement complex behaviors, and by the periodic task that manages the communication with the other robots through the wireless interface. All processes access the RTDB with local non-blocking function calls, `DB_put()` and `DB_get()` that allow writing and reading records, respectively (Fig. 2 shows the prototypes of the RTDB related function calls). `DB_get()` further requires the specification of the agent from which the item to be read belongs to, in order to identify the respective area in the database.

```
int DB_init (void)
void DB_free (void)
int DB_put (int _id, void *_value)
int DB_get (int _agent, int _id, void *_value)
```

Fig. 2. The RTDB interface function calls

B. Internal Structure

The RTDB is organized in a set of records plus a set of associated data blocks. The records contain the fields referred in Fig. 3, namely an identifier, a pointer to the respective data block, the size of that block, a timestamp for computing the age of the data, the update period reflecting the dynamism of the respective item, and a control field for data consistency.

To enforce data consistency during concurrent accesses a double data block is used for each record. With this scheme any write operation on that item is made on the block that is free at that instant. This method ensures consistent data retrieval, as long as there is only one process updating the same item.

```
typedef struct {
  int id;
  int offset;
  int size;
  timeval timestamp;
  int period;
  int read_bank;
} TRec;
```

Fig. 3. The fields of a generic RTDB record

C. Interface with the communication protocol

The RTDB is like a dual-port memory in which the application, either producer or consumer, uses one side and the communication protocol uses the other side. On the application side, the accesses to the RTDB are carried out on an item basis, independently from each other. However, the communication protocol works in cycles that are approximately periodic and in each cycle each robot in the team transmits once, only (see section IV). Therefore, to reduce communication overhead, when the time comes for a robot to transmit, the process that manages the communication protocol piggybacks all items that are to be transmitted at that point and assembles them in the required number of packets, typically just one, which are then dispatched to the wireless medium.

D. Age of the data

A time stamping mechanism allows the application to estimate the age of the data and thus detect situations of loss of temporal validity. However, for the sake of simplicity, this middleware does not include a global clock service and thus, the clock in each robot is not correlated. To circumvent such difficulty, the middleware computes time intervals, only.

When a producer writes a datum in the RTDB its local time $t1$ is saved in the `timestamp` field (see Fig. 4). Later on, when the communication protocol fetches the datum to transmit it over the wireless interface, it computes the difference between the current local time $t2$ and the saved timestamp, which is the age of the datum at the time of transmission (producer side). The calculated datum age $t2-t1$ is attached to the datum itself and transmitted together in the same packet.

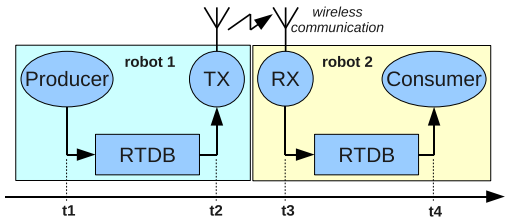

Fig. 4. Datum age calculation

When a packet is received by the communciation system at the consumer side, each datum is written in the RTDB and the corresponding data age, from the producer, subtracted from the current local time $t3 - (t2 - t1)$ and saved in `timestamp`. When a consumer process retrieves an item from the RTDB, the difference from the current time to the time stamp saved in `timestamp` is computed, which is an estimate of the age of the data, from the moment it was produced to the moment in which it was consumed.

This estimation, however, does not account for the transmission time, which depends on the actual bit rate, on the latency to access the channel and on possible retransmissions. However, as shown further on in section VI, the communication protocol used together with this middleware has a positive impact on the transmission time, leading to a relatively constant latency that can be easily added to the age estimation to improve its accuracy.

E. Configuration

The configuration of the RTDB is done automatically by a parser, based on a text description file that specifies the team definition and the RTDB composition (Fig. 5). It identifies each `AGENT` (robot) in the team by a name and each data `ITEM` is defined with an `id`. To compute the data size, the user must introduce the name of the item, `dataname`, and the `headerfile` where it is defined. Also, an optional `period` value can be specified, which sets the data refresh period of the respective item in number of communication cycles (see section IV). The `SCHEMA` allows indicating for each data item `id` whether it is shared or local, the shared ones being disseminated by the communication protocol. The `ASSIGNMENT` clause associates one or more `SCHEMA` to one or more `AGENTS`.

```
AGENTS = ag1, ag2;
ITEM id1 { dataname; headerfile; period; }
SCHEMA sh1 { shared = id1, id2;
             local = id3, id4; }
ASSIGNMENT { schema = sh1; agents = ag2; }
```

Fig. 5. The RTDB configuration file

The RTDB middleware easily accommodates the coexistence of heterogeneous robots. However, the team composition as well as the RTDB configuration must currently be known before run time. The dynamic configuration of the RTDB will be addressed in future work.

IV. THE COMMUNICATION PROTOCOL

The basis of the communication protocol is a Time-Division Multiple-Access (TDMA) cycle with a sequece of slots, each allocated to a single robot. However, since the load in the network cannot be totally controlled by the team, the only alternative left is to adapt to the current channel conditions and reduce access collisions among team members. This is achieved using an adaptive TDMA transmission control as proposed in [21], which synchonizes the transmissions within the team based on the actual reception instants of the respective packets. The TDMA round period is set off-line and called *team update period (T_{tup})*, determining the responsiveness and the temporal resolution of the global communication. It is, thus, set according to the application requirements. T_{tup} is divided equally by the number of team members, N, generating the TDMA slot structure, with equal

slots of duration T_{xwin}. The agents transmit at the begining of their slots and thus their transmissions are separated as much as possible.

When using a fixed number of team members, if some robots are not active at a given time, e.g., because of a crash or unavailability, the values of T_{xwin} will be smaller than needed, i.e., the slots are unnecessarily short since some of them are not used. Note that a smaller T_{xwin} reduces the leeway to accomodate delays caused by the uncontrolled traffic and thus increases the probability of loss of synchronization and of collisions within the team. Therefore, a self-configuration capability was added to the protocol, to cope with variable number of team members [5]. This mechanism supports the dynamic insertion / removal of robots in the protocol in a fully distributed way. Currently, the T_{tup} period is still constant but it is divided by the number of running agents at each instant, designated K, with $K \leq N$, maximizing the inter-transmission separation between agents.

V. THE PMAN

The middleware presented in this work is based on Linux, which is a general propose operating system (GPOS) thus having the advantage of supporting a variety of hardware and providing several development environments. Conversely, as any kind of GPOS, it has a limited support to the temporal management of the applications. Therefore, the RTDB middleware was complemented with the PMan process manager [6], which offers a set of time-related services to support the development of real-time applications over Linux. The PMan services include:

- Automatic activation of recurrent tasks;
- Settling of relative phase control, allowing to establish temporal offsets among tasks;
- Precedence constraints, conditioning the release of processes to the conclusion of a set of predecessors;
- On-line process management and QoS adaptation, allowing adding and removing processes at run-time as well as changing dynamically the temporal properties of the executing ones, without service disruption.

Particularly, enforcing precedence constraints and setting relative offsets is quite helpful for the programmer. In fact, they are applied simply by adequate tasks configuration when they are set up, without the need to use any synchronization primitives, thus simplying the application development, particularly in complex platforms such as multi-core processors.

VI. TIMING ISSUES

The communication system presented in section IV is not synchronized with the control system of the robots, including the tasks managed by the PMan, due to the adaptive nature of the protocol that keeps changing its cycle duration. This may lead to extra delays in the refreshing of the remote data that the programmer must be aware. In particular, when a robot accesses a local image of a datum from other team member, that datum could be as old as:

$$max\_data\_age = min(T_{rcpp}, T_{dup} * T_{tup}) + T_{wt} + (T_{dup} * T_{tup})$$

This worst case data age corresponds to when the communication system fetches the data in the producer for transmission just before that data being updated by the control process in the respective robot. Thus, at that point, that data can be as old as one period of the respective producer (T_{rcpp}). However, this latency cannot be larger than the data update period configured in the RTDB ($T_{dup} * T_{tup}$) thus, the minimum of the two must be considered. Note that T_{dup} is the refresh period in integer number of communication cycles, i.e., team update periods (T_{tup}). The transmission of the data over the air takes some time that must also be accounted for (T_{wt}). Finally, when the consumer accesses the data on its side, the data can be waiting in the respective buffer for at most another data update period ($T_{dup} \times T_{tup}$).

Within the above expression, only the wireless transmission delay is unknown and it is certainly variable with the traffic load in the network. To evaluate the wireless transmission delay (T_{wt}) we carried out a few experiments with and without traffic interference and with and without the Reconfigurable and Adaptive TDMA protocol, to help assess its impact in this parameter. Fig. 6 shows the histograms of the transmission delay using packets with 379 bytes. The network was configured in managed mode, all the transmissions were carried out by the Access Point (AP) and the multicast packets were transmitted at a rate of 1Mbps. The experiments used 4 robots with a T_{tup} of 50 ms and logs were extracted for about 9 minutes of continued operation. The interference traffic was generated by an external laptop pinging the AP using 1000 bytes packets at a rate of 5 and 10 ms in two different experiments. All robots were started at the same time using a trigger signal generated by the external laptop, which causes a situation of maximum contention among the team members.

Fig. 6.a) and Fig. 6.b) show the transmission delay for the cases of using and not using the RA-TDMA protocol, respectively. It is clear that, in the former case, the synchronization imposed by the protocol immediately sorts out the high contention caused in the starting period and the interference among teams members is practically eliminated. Without synchronization, the team members continue interfering with each other, leading to a substantial increase in the transmission delay. Moreover, the impact of the interfering traffic is also worse without synchronization than when using the RA-TDMA protocol, showing the benefit of using this middleware. Finally, it is also important to note that the high contention provoked in the experiments represents a real situation since when the clocks are not synchronized, the phase drifts will cause situations of similar high contention, which last for several minutes of operation, leading to periods of degraded communication performance.

VII. CONCLUSIONS

This papers describes a novel middleware for teams of mobile robots that relies on a real-time database partially replicated, containing both local and remote state variables, in a distributed shared memory style. The remote data is updated autonomously in the background by a WiFi-based

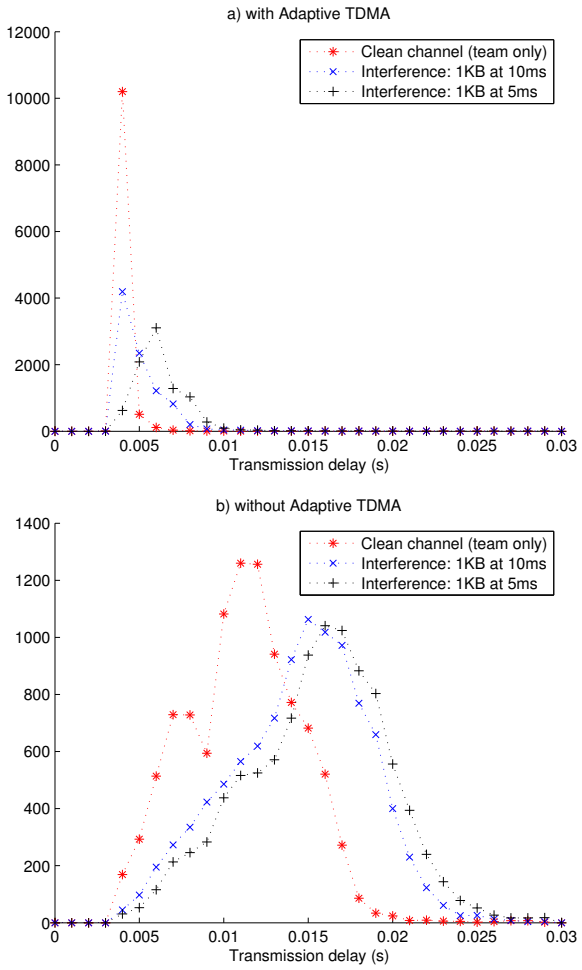

Fig. 6. Transmission delay. a) using proposed protocol; b) without protocol

wireless communication protocol, at an adequate refresh rate, using a modified TDMA scheme, that self-adapt to the current channel conditions and the number of team robots. The software infrastruture is complemented with a task manager that provides scheduling and synchronization services to the application processes on top of the Linux operating system, supporting offsets and precedence constraints without need for explicit synchronization primitives.

This paper focused on the existing middlewares to support cooperation among robots, putting the proposed middleware in context, highlighting its advantages. It is an affordable and dependable middleware that facilitates the programming of the cooperative applications, not needing explicit use of communication, thus abstracting away distribution. Moreover, a novel timestamping mechanism provides enhanced information on the data age at the time of consumption. The worst-case age of the remote data was also analyzed and a bound was given. This age depends on the transmission latency. Several experiments were then carried out to assess such latency. The results show the positive impact of the synchronization embedded in the middleware that avoids interference among team members.

VIII. ACKNOWLEDGMENTS

This work was partially supported by the European Comission through grant ArtistDesign ICT-NoE-214373 and Portuguese Government through grant FCT - SFRH/BD/29839/2006.

REFERENCES

[1] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.
[2] N. Mohamed and J. Al-Jaroodi, "Characteristics of Middleware for Networked Collaborative Robots," in *Proc. of the International Symposium on Collaborative Technologies and Systems*, Irvine, USA, 2008.
[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup: The Robot World Cup Initiative," in *Proc. of the IJCAI-95 Workshop on Entertainment and AI/Alife*, Montreal, August 1995.
[4] L. Almeida, F. Santos, T. Facchinetti, P. Pedreiras, V. Silva, and L. S. Lopes, "Coordinating Distributed Autonomous Agents with a Real-Time Database: The CAMBADA Project," in *ISCIS*, ser. Lecture Notes in Computer Science, C. Aykanat, T. Dayar, and I. Korpeoglu, Eds., vol. 3280. Springer, 2004, pp. 876–886.
[5] F. Santos, L. Almeida, and L. S. Lopes, "Self-configuration of an Adaptive TDMA wireless communication protocol for teams of mobile robots," in *Proc. of the IEEE International Conference on Emerging Technologies and Factory Automation*, Hamburg, Germany, 2008.
[6] P. Pedreiras and L. Almeida, "Task Management for Soft Real-Time Applications based on General Purpose Operating Systems," in *Proc. of the 9th Workshop on Real-Time Systems*, Belem, Brazil, May 2007.
[7] R. Hafner, S. Lange, M. Lauer, and M. Riedmiller, "Brainstormers Tribots Team Description," in *RoboCup TDP*, Suzhou, China, 2008.
[8] O. Zweigle, U.-P. Kappeler, T. Ruhr, K. Hussermann, R. Lafrenz, F. Schreiber, A. Tamke, H. Rajaie, A. Burla, M. Schanz, and P. Levi, "CoPS Stuttgart Team Description," in *RoboCup TDP*, Atlanta, 2007.
[9] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and R. Reddy, "The HERSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, pp. 213–253, 1980.
[10] S. Enderle, H. Utz, S. Sablatnog, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for Autonomous Mobile Robots," in *Proc. of the IFAC Conference on Telematics Applications in Autonomous and Robotics*, Weingarten, Germany, July 2001.
[11] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro: Middleware for Mobile Robot Applications," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 493–497, August 2002.
[12] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and System*, Alberta, Canada, August 2005.
[13] J. Yoo, S. Kim, and S. Hong, "The Robot Software Communications Architecture (RSCA) QoS-Aware Middleware for Networked Service Robots," in *Int. Joint Conf. SICE-ICASE 2006*, Busan, Korea, 2006.
[14] A. Corsaro, L. Querzoni, S. Scipioni, S. T. Piergiovanni, and A. Virgillito, "Quality of Service in Publish/Subscribe Middleware," R. Baldoni and G. Cortese, Eds. IOS Press, 2006.
[15] "Data Distribution Service for Real Time Systems, v.1.2," *OMG*, 2007.
[16] D. Calisi, A. Censi, L. Iocchi, and D. Nardi, "OpenRDK: A Modular Framework for Robotic Software Development," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, September 2008.
[17] D. Miorandi and S. Vitturi, "Performance analysis of producer/consumer protocols over IEEE802.11 wireless links," in *Proc. of the IEEE Int. Workshop on Factory Communication Systems*, 2004.
[18] V. Milutinovic and P. Stenstrom, "Special Issue on Distributed Shared Memory Systems," in *Proc. of the IEEE*, vol. 87, no. 3, March 1999.
[19] P. Baer, R. Reichle, M. Zapf, T. Weise, and K. Geihs, "A Generative Approach to the Development of Autonomous Robot Software," in *Proc. of the EASe'07 - 4th IEEE Int. Workshop on Engineering of Autonomic and Autonomous Systems*, Tucson, USA, March 2007.
[20] H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer Academic Pub, 1997.
[21] F. Santos, L. Almeida, P. Pedreiras, L. S. Lopes, and T. Facchinetti, "An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication among Mobile Autonomous Agents," in *Proc. of the WACERTS04 Workshop on Architectures for Cooperative Embedded Real-Time Systems*, Lisbon, Portugal, December 2004.