Software Engineering for Experimental Robotics

Springer Tracts on Advanced Robotics

Edited by Davide Brugali

Part I

Control Architectures and Toolkits

MRT: Robotics Off-the-Shelf with the Modular Robotic Toolkit

Andrea Bonarini, Matteo Matteucci, and Marcello Restelli

Politecnico di Milano, Department of Electronics and Information, Milan, Italy
{bonarini,matteucci,restelli}@elet.polimi.it

1 Introduction

Since robotic applications are becoming more and more sophisticated, the design, development, and maintenance of the related software may benefit from a modular approach both in terms of flexibility and reusability. By decomposing a complex system into several simpler functional modules, it is possible to separate responsibilities and parallelize efforts. Many robotic systems, even operating in really different application domains, share several functionalities. The use of a modular approach allows to reuse the same functional units in different applications, thus reducing the development time and increasing the software reliability.

In our bi–decennial experience in making mobile robots, we have identified a certain amount of functionalities that are involved to perform tasks autonomously. We have devised a modular approach to their implementation, based on the principle that each specific module can interact with others by simply exchanging messages in a distributed setting. Within this framework, different modules may even run on different machines, and data may be integrated by modules providing aggregated information to the others. Moreover, modules managing the interaction with the environment are designed to make independent the actual interface with physical devices from the needed data processing. A middleware is also provided to make the interaction among modules transparent with respect to their physical distribution. This makes also possible to implement multi–agent and multi–sensor systems integrated in a unique network, which may change its structure in time.

The following section describes the main functionalities we have identified for autonomous robots applications, and how they interact. We have implemented modules for all these functionalities in the Modular Robotic Toolkit (MRT). In Section 2, we summarize the main features of each of our modules, and we present DCDT, our middleware to support interaction among them. MRT has proved effective to provide the functionalities needed for different successful applications. In Section 4, we present how some MRT modules

have been integrated in these applications: Roby, a robot able to explore hostile environments, FollowMe, a guide for indoor environments, and the Milan RoboCup Team, a robotic soccer team participating to the RoboCup [14] competition.

2 Functional modules and their interaction in autonomous robots

Recently, the interest in modular architectures is growing in the robotic field [7], with the final aim of producing frameworks where a set of off-the-shelf modules can be easily combined and customized to implement robotic applications with minimal effort and time. All the proposed approaches consist of the definition of basic modules and an integration layer, which often relies on commercial standards (e.g. CORBA, JINI, etc.). In the following, we present a general functional decomposition of autonomous robot applications and we motivate our choice of adopting the publish/subscribe paradigm to support interaction.

2.1 Functional Modules

In a robotic application, the typical functional modules can be classified into three main categories: sensing modules, reasoning modules, and acting modules. Sensing modules are directly interfaced with physical sensors (e.g., sonars, laser range finders, vision systems, encoders, bumpers, gyroscopes), with the aim of acquiring raw data, processing them, and producing higher level information for the reasoning modules.

Acting modules are responsible for controlling actuators, implementing the decisions of reasoning modules. Thus, the reasoning modules need to know neither which sensors, nor which actuators are actually mounted on the robot; this allows their reuse in different contexts. We decompose each sensing and each acting module into two sub-modules: the *driver*, which directly interacts with the physical device, and the *processing* sub-module, which is interfaced on one side with a driver, and on the other with some reasoning modules or the world model. So, thanks to its driver, a processing sub-module may abstract from the physical characteristics of the specific sensor/actuator, and it can be reused with different devices of the same kind. For instance, let us consider a mobile robot equipped with a firewire camera. The driver should implement some functionalities such as the interface for changing the camera settings and the possibility to capture the most recent frame. The processing sub-module should extract from the acquired image the most relevant features, which can be used by reasoning modules like the ones devoted to localization, world modeling, and planning. If we decide to change the firewire camera with an USB or an analog camera, or if we have several robots equipped with different

cameras, we may have to re-implement the driver sub–module, but we can reuse the same processing sub–module.

Reasoning modules represent the core of each robotic software, since they code the decision–making processes that determine the robot behavior. In MRT, reasoning modules are in principle independent from sensors that have been used in the specific application. Thus, all the information gathered through different sensors can be integrated into a world representation, on which the other modules may perform their inferential processes. The outcomes of reasoning modules are high level commands to be processed by acting modules, and then executed by actuators. Most real–world robot applications can be built using some of the following modules:

- *localization module*: it has to estimate from sensor data the robot pose with respect to a global reference frame, using a map of the environment;
- *world modeling module*: it builds and maintains a representation of the external world, integrating the information gathered through sensors and, eventually, the interaction with other robots;
- *planning module*: based on knowledge about the environment and the robot itself, this module selects the most appropriate actions to reach the given goals, while respecting task constraints;
- *sequencing module*: this module is responsible for the execution of a given plan, monitoring its progress, and handling exceptions as they arise;
- *controlling module*: it contains all the control laws, typically implemented as reactive behaviors [8], to decide which actions the robot has to execute;
- *coordination module*: multi-robot applications often require a module that allows robots to share perceptions and to communicate intentions in order to perform effective task allocations.

Figure 1 depicts the robotic functional modules described in this section, arranged in a typical hybrid control architecture [11], in which the deliberative and reactive layers are combined through a middle layer that implements a sequencing mechanism. At this point it is worth noticing that a key factor in software reuse is the configurability of these modules. In fact, it is really hard to build general modules that can be usefully employed in different applications, unless it is possible to configure them to fit specific needs. For each module, it is important to identify the application specific parameters, so that they can be easily specified on a case by case basis. According to this view, the development process of each robotic application should consist of: implementation of the driver sub-modules for each sensor/actuator device, selection and parametrization of the reasoning modules required by the application.

2.2 Publish/Subscribe Middleware for Interaction

Having different modules that cooperate to obtain a complex control architecture shifts the focus from the classical monolithic approach to a more flexible distributed approach. This shift in perspective introduced by programming

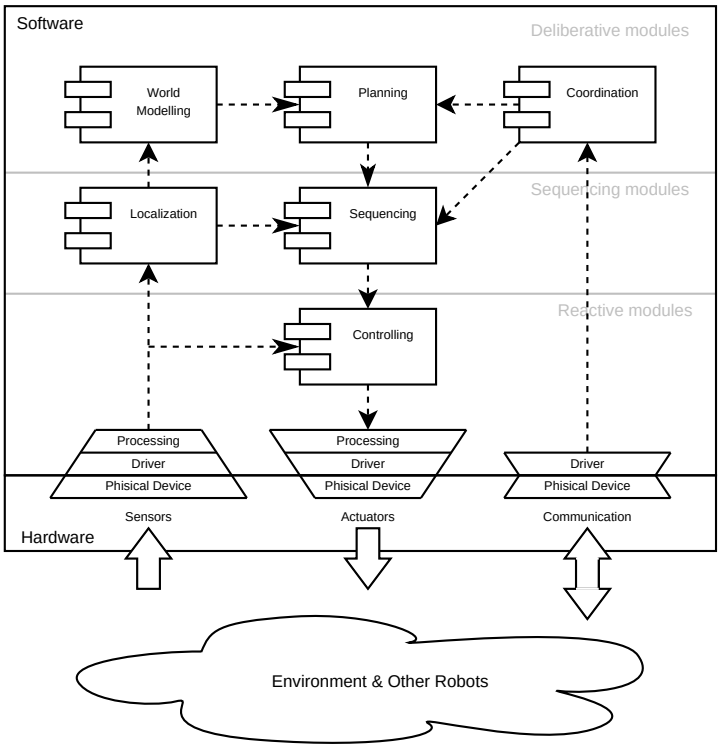

Fig. 1. The MRT general architecture.

in a distributed, concurrent, and modular environment has been clearly described in [9], and the adoption of an integration middleware becomes a key for the effectiveness of this approach. At the same time, autonomous robotic systems, particularly those involving multiple robots and environmental sensors, are becoming increasingly distributed and networked; thus, people involved in robotics are getting conscious that developing multirobot systems might require tools supporting integration and communication [12].

No definitive answer has been given about which is the proper middleware for integration, but a key issue to be analyzed in making this choice is certainly how data are exchanged. In the past we have proposed the publish/subscribe model as the most interesting communication model in this new development scenario providing a compared analysis of the publish/subscribe middleware tools currently used in robotic applications [15]. While the publish/subscribe approach is not the only approach that can be used for integration in robotics (among others are: client/server, distributed blackboards [13] or request/response models [18]), it has been widely used for this purpose as testified by [16].

In the publish/subscribe model, data providers publish data and consumers subscribe to the information they require, receiving updates from the providers as soon as this information has been published. This paradigm is based on the notion of event or message. Components interested in some class of events subscribe expressing their interests. Components providing information publish it to the rest of the system as event notification. This model introduces a decoupling between producers and consumers since publishers and subscribers do not know each other. This model offers significant advantages in situations where data transferred correspond essentially to time-changing values of an otherwise continuous signal (e.g., sensed data, control signals, etc.). A single subscription replaces a continuous stream of requests and the data is transferred with minimum delay since the exchange is one way and asynchronous. The publish/subscribe model is also notification-based and this is very beneficial when a system needs to monitor a great number of perhaps infrequent events, while in a client/server model (or shared memory model), the monitoring process must continuously poll for possible changes. One-to-many communication is supported by the publish/subscribe model in a natural way, and its implementation can often take advantage of multicast and broadcast mechanisms at the network level to improve the efficiency of event notification.

3 The Modular Robotic Toolkit

Our research effort in modular software for robotics has its roots in the development of autonomous systems in an academic environment, where generations of students have to face necessarily limited parts of the whole robot development, and they should transfer their knowledge about the evolving robots to the next generations. The modular approach began to become effective with the experience made developing the Milan RoboCup Team (MRT) [2], a team of soccer robots for the RoboCup competition. This effort has resulted in a more general Modular Robotic Toolkit (MRT again) that we are presenting in this paper and that has been used to develop other robotic applications in different domains, thus experiencing the benefits of a modular approach.

3.1 MRT Modules

We have implemented all the reasoning modules described in Section 2 and, according to the specific application, we have selected and customized the required modules, as discussed in Section 4. In the following, we describe each module and the publish/subscribe integration middleware developed. In this section, we describe each module and in the following one the publish/subscribe integration middleware developed.

MUREA: the localization module

We have adopted *MUREA* (MUlti-Resolution Evidence Accumulation) [17], a mobile robot localization algorithm for known 2D environments. The localization space is divided into subregions (cells) and then an evidence accumulation method is applied, where each perception votes those cells that are compatible with the map of the environment. In order to reduce the complexity of working with a fine grid, we have adopted a multi–resolution scheme. We start applying evidence accumulation on a coarse grid, with few large cells. Then, we select and refine (i.e., divide into smaller cells) those cells that have collected highest votes. This module has several configurable parameters that allow its reuse in different contexts. The map of the environment is specified through a configuration file in which are listed the elements that can be perceived by the robot. Each element is defined by a name, its geometrical shape (point, segment, circle, etc.), and a list of attributes that describe the element. Other parameters that can be set in the configuration file are the range of feasible positions, the required accuracy, and a timeout. In order to grant the reusability in different robotic applications, MUREA completely abstracts from the sensors used for acquiring localization information, since its interface relies on the concept of *perception*, which is shared with the processing sub-module of each sensor.

MAP: the world modelling module

The world modelling module is a nodal point for many robotic applications. We have studied and proposed *MAP* (Map Anchors Percepts) [3], a module that builds and maintains in time, through an anchoring process, the environment model on the basis of the data acquired by sensors. MAP contains a hierarchical conceptual model, which must be specified for each application, where the classes of objects that can be perceived, and their attributes, are defined. The MAP module is divided into three sub-modules. The *Classifier*, that receives from sensors descriptions of perceived objects in terms of percepts (symbolic features), and identifies, for each perceived object, the concept that has the best matching degree. The classification process generates conceptual instances with an associated degree of reliability. These are passed to the *Fusion* sub-module. Since any physical object may be perceived at the same time by distinct sensors, the output of the Classifier may contain several conceptual instances that are related to the same physical object. The goal of the Fusion sub-module is to merge the conceptual instances that are supposed to be originated from the perception of the same object. Finally, Map performs the tracking phase, which consists of maintaining in time a coherent state of the model and a correct classification of the instances. The *Tracker* tries to match the conceptual instances perceived in the past with those generated by the latest perception. Then, the dynamic properties of the conceptual instances are updated through Extended Kalman filtering.

This approach is well suited also for multi–robot domains, where each robot may take advantage of the data acquired by its teammates. Each agent can be seen as an intelligent sensor that, instead of producing symbolic features, generates conceptual instances. These can be processed directly by the Fusion sub-module of any other agent that shares the same hierarchical conceptual model.

SPIKE: the planning module

Trajectory planning is obtained by *SPIKE* (Spike Plans in Known Environments) a fast planner based on a geometrical representation of static and dynamic objects in the environment. The environment to navigate, in SPIKE, is modeled as a 2D space, the robot is considered as a point with no orientation, and static obstacles (e.g., walls, bins, furniture) are described using basic geometric primitives such as points, segments and circles. SPIKE exploits a multi-resolution grid over the environment representation to seek a proper path from a starting position to the requested goal; this path is finally represented as a polyline that does not intersect obstacles. Moving objects in the environment can be easily introduced in SPIKE representation of the environment as soon as they are detected and they can be considered while planning. Finally doors or small (w.r.t. the grid resolution) passages can be manged by the specification of *links* in the static description of the environment. The resolution of the plan is customizable and computation simply requires the description of the environment, the starting point and the goal point; the output is given as a sequence of positions the robot has to reach from the starting point to the goal. This trajectory is computed by using an adapted A^* algorithm on the multi-resolution grid, and then optimized by using geometrical considerations. The grid representation, inspired by the work from Sven Behnke [1], uses a coarse resolution for the (mostly free) area to be navigate, and a finer resolution near static objects and around the robot to manage, respectively, small passages and dynamic obstacles. Path computation as well as the resolution of the grids used can be easily customized by considering robot size, required accuracy, and a safety distance from obstacles.

SCARE: the sequencing and coordination module

In our applications, the sequencer and the multi-robot coordination functionalities are implemented as two sub-modules of a module called *SCARE* (Scare Coordinates Agents in Robotic Environments) [6]. Through a two-phase distributed process, SCARE assigns activities to each team member. Activities may belong to one of two categories: *jobs* that are single robot activities, and *schemes*, multi-robot activities ruling the performance of a synchronized sequence of jobs by a team of robots. In the *decision making* phase, SCARE gathers up (through sensors and, in case, communication with other robots) all the available information about the current situation, and evaluates for

each activity several parameters such as the physical attitude (e.g., how much the robot is *physically suited* for the job), the *opportunity* (e.g., how much, in the current situation, the robot is suited for the job), and the *need* (e.g., how much, in the current situation, the job is useful for the team). By employing multi-objective analysis techniques, SCARE produces for each robot an ordered list of activities called *agenda*. In the *coordination* phase, SCARE searches for the best job allocation to agents by observing coordination constraints such as *cardinality* (the minimum and maximum number of robots that can be assigned to the activity at the same time) and *scheme coverage* (a scheme can be assigned only if there is an appropriate robot for each functionality). Once the job allocation has been performed, the sequencing sub-module decomposes the assigned job into simpler subjobs, schedules their execution, and monitors the environment to react to unexpected events.

MrBRIAN: the control module

Our control module is *MrBRIAN* (Multilevel Ruling Brian Reacts by Inferential ActioNs) [4], a fuzzy behavior management system. Behaviors are implemented as set of fuzzy rules whose antecedents match context predicates, and consequents define actions to be done. Behavioral modules are activated according to the CANDO conditions defined for each of them as fuzzy predicates. Actions are proposed by each behavioral module with a weight depending on the degree of matching. The proposals are then composed with weights coming from the evaluation of WANT fuzzy predicates, defined for each behavioral module to take into account the desirability to apply a given module in a given situation. Among the WANT conditions we have context conditions, which are evaluated on the information coming about the environment and define the opportunity to apply a behavior in a given situation, and coordination/planning conditions, eventually set by other modules of the architecture, to model the opportunity to apply a behavior given intra-agent plans and coordination needs.

In MrBRIAN, it is also possible to organize behaviors in a hierarchy. It this case, behaviors at the higher levels match also the action proposed by others with lower priority, and may decide to inhibit them, either partially or completely. MrBRIAN is completely configurable for the specific application, by specifying which behaviors are required, their implementation, their priority, their activation conditions (CANDO), and desirability conditions (WANT), and the fuzzy sets involved in the definition of predicates.

3.2 DCDT (Device Communities Development Toolkit)

To integrate modules in MRT we use *DCDT* (Device Community Development Toolkit) [10], a publish/subscribe middleware able to exploit different physical communication means in a transparent and easy way. DCDT is a multi-threaded architecture that consists of a main active object, called Agora,

hosting and managing various software modules, called *Members*. Members are basically concurrent programs/threads executed periodically or on the notification of an event. It is possible to realize distributed applications running different Agoras on different devices/machines, each of them hosting many Members. It is also possible to have on the same machine more than one Agora (hosting its own Members), to emulate the presence of different robots without the need of actually having them connected and running. Agoras on different machines are able to communicate with each other through particular members, called *Finder*, which are responsible for finding each other dynamically with short messages via multicast. The peculiar attitude of DCDT toward different physical communication channels (e.g., RS-232 serial connections, USB, Ethernet or IEEE 802.11b , etc.) is one of the main characteristics of this publish/subscribe middleware.

DCDT messages are characterized by header and payload fields. In the header we have: the unique identifier of the message type (i.e., the content/subject of the message), the size of the data contained in the payload and some information regarding the producer (e.g., producer id, time of construction, etc.). Members use unique identification types to subscribe and unsubscribe messages available throughout the community. Messages can be shared basically according to three modalities: without any guaranty (e.g. UDP), with some retransmissions (e.g. UDP retransmitted), or with absolute receipt guaranty (e.g. TCP).

3.3 MRT Messages

In MRT, the interfaces among different modules are implemented by exchanging messages. According to the publish/subscribe paradigm, each module may produce messages that are received by those modules that have expressed interest in them. In order to grant independence among modules, each module knows neither which modules will receive its messages nor the senders of the messages it has requested. In fact, typically, it does not matter which module has produced the acquired information, since modules are interested in the information itself. For instance, the localization module may benefit from knowing that in front of the robot there is a wall at the distance of 2.4 m, but it is not relevant which sensor has perceived this information or whether this is coming from another robot.

Our modules exchange XML (eXtensible Markup Language) messages whose structure is defined by a shared DTD (Document Type Definition). There are several motivations for our choice of using XML messages, instead of messages in a binary format. Since XML messages are in text format, they can be directly read by humans and can be edited by any text editor. The use of DTDs allow to specify the structure of each message and check that the messages received are well-structured. Furthermore, XML messages can be easily changed and extended. Finally, the input interface for XML messages is greatly simplified by the existence of standard modules for syntactic

```
<message robot="Roby" module="Vision" data="Perceptions"
         timestamp="11982374383" />
  <object name="RedObject0" class="Object" />
    <attribute name="Color"    value="Red" />
    <attribute name="Distance" value="332" std_dev="10" rel="1"  />
    <attribute name="MinAngle" value="175" std_dev="2"  rel="0.8"/>
    <attribute name="MaxAngle" value="185" std_dev="2"  rel="0.9"/>
  </object>
  <object name="WhiteGreenWhite0" class="Transition" />
    <attribute name="Color"    value="WhiteGreenWhite" />
    <attribute name="Distance" value="505" std_dev="24" rel="0.7"/>
    <attribute name="Angle"    value="245" std_dev="1"  rel="0.9"/>
  </object>
  <object name="YellowObject0" class="Object" />
    <attribute name="Color"    value="Yellow" />
    <attribute name="Distance" value="178" std_dev="10" rel="1" />
    <attribute name="MinAngle" value="275" std_dev="2"  rel="1" />
    <attribute name="MaxAngle" value="331" std_dev="2"  rel="1" />
  </object>
</message>
```

Fig. 2. Example of an XML message used in the MRT framework

parsing. These advantages are partially offset by some drawbacks: the amount of transferred data typically increases, parsing may need more time, and binary data can not be included directly in the message. However, none of these drawbacks has been a limitation in our applications.

Each message contains some general information (e.g., the time-stamp) and a list of objects each characterized by a name and its membership class. For each object it is possible to define a number of attributes, that are tuples of name, value, and, optionally, variability and reliability. In order to correctly parse the content of the messages, modules share a common ontology that defines the semantic of the used symbols. In Figure 2 we have reported an example of XML message produced by the vision sensor module.

4 Applications

We have used the MRT framework to develop robotic applications with different robot platforms and in a few contexts. The general structure of the Modular Robotics Toolkit is reported in Figure 3 including all the implemented modules and the typical message passing connections between them.

Since we have used different robotic platforms, from custom bases with differential drive or omnidirectional wheels to commercial all-terrain platforms, the use of a proper abstraction layer in sensing and actuation modules allowed us to focus mainly on the development of robot intelligence.

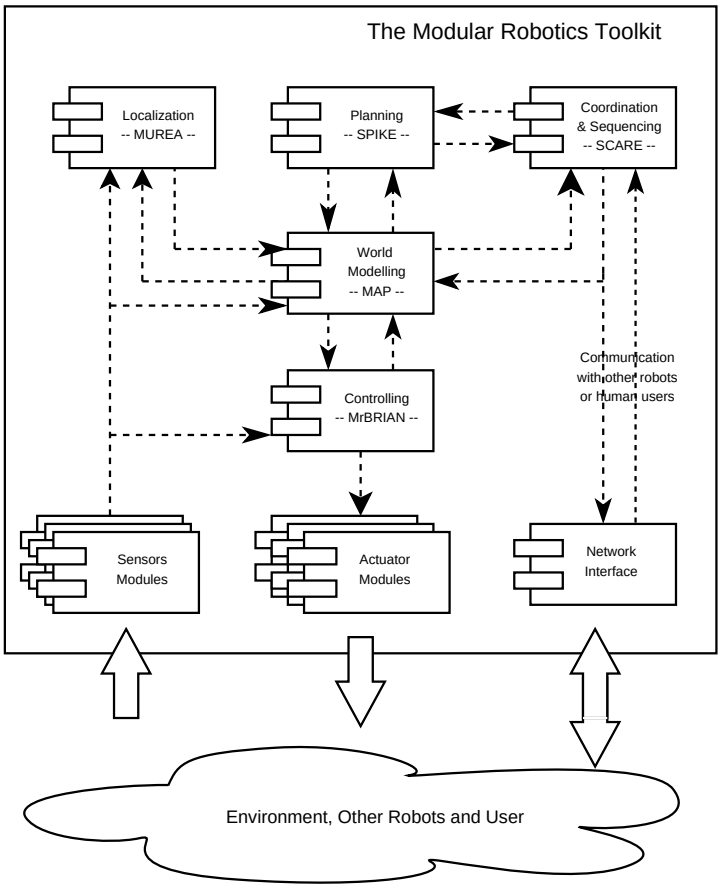

Fig. 3. The general architecture implemented by the Modular Robotics Toolkit; all the reasoning modules are present and typical message passing paths are reported using dashed arrows.

An application might not need all of the modules implemented in MRT, and the modular structure makes it possible to integrate also only a subset of the available modules. In the following, we present three different architectures implemented with MRT to solve specific tasks. We are just interested in presenting three case studies where MRT was successfully used to reduce in a sensible way the "time to market" for the requested application, thanks to its modularity.

4.1 Roby

ROBY is a project (in the framework of the European project GALILEO) that we have developed in collaboration with the Italian company InfoSolution[1]. In this project, an ActiveMedia P3-AT platform with a differential drive kinematics, equipped with a Differential GPS device, must travel between two points, chosen over a pre-defined map of an outdoor environment. A path planner, different from SPIKE, was already available and run on a machine not on the robot. It computed the sequence of points that describe a free path, and sent the set of way points to the robot through a wireless connection.

The map of the planner contains only information about static, known objects (e.g., buildings, bridges, etc.); for this reason, the robot is also equipped with a sonar belt in order to detect obstacles, so that it can manage situations in which the trajectory produced by the planner is obstructed by something, possibly not included in the map (e.g., cars, trees, people, etc). In Figure 4 we show the architecture of the ROBY case study. Only few modules of the general architecture are used to implement a path-following behavior. In this case, MRT modularity has been exploited for the seamless integration in the control architecture of a planner developed by an external contractor.

In this application, no complex world modelling is needed since the differential GPS already provides a good estimation of the robot position and this is sufficient to accomplish the task. Even if this is a single robot domain, the sequential functionality, implemented in SCARE, has been used to enable MrBRIAN to follow the sequence of path–points with a simple reactive behavior. Each job `ReachPathPointJob` terminates when either the robot reaches the related path point (success condition), or a timeout expires (failure condition). In the former case, the current job ends, and the `ReachPathPointJob` job related to the next path point is activated. In the latter case, the whole task is aborted, and the planning module is requested to produce, starting from the current robot position, a new sequence of path points.

During navigation, data collected from the sonar belt are used directly by MrBRIAN to feed the reactive `AvoidObstacle` behavior, since we want our robot to promptly react in avoiding collisions. This behavior has a higher priority w.r.t. `ReachPathPoint`, so that when and obstacle is faced, `AvoidObstacle` tries to avoid it by leaving as less as possible the path that `ReachPathPoint` is suggesting to follow.

Although the implemented structure is very simple, we have obtained satisfactory results in several trials with different conditions, and the robot was always able to reach the goal point dealing with unforeseen obstacles. The wireless connection has been also used by a person to drive the robot in a teleoperated fashion while keeping active sensing modules to implement safety obstacle avoidance.

[1] Sample movies on the real robot are available on the at URL `http://www.infosol.it/Movies/offer_roby_movies.htm` on the InfoSolution web site.

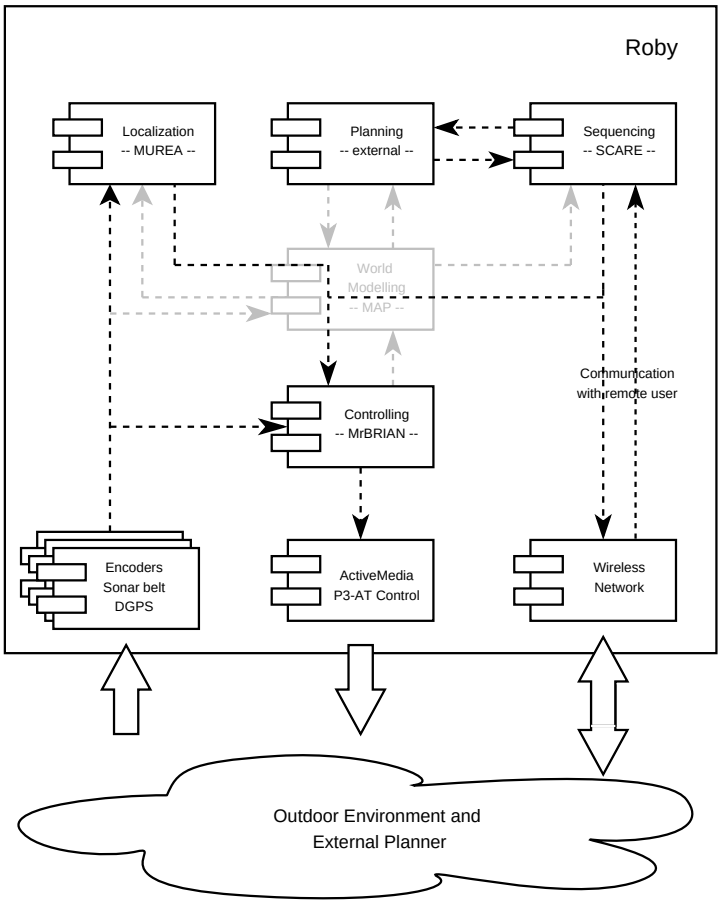

Fig. 4. MRT modules involved in the ROBY case study.

4.2 FollowMe

FOLLOWME is a project to develop a guide robot to be deployed in a museum. In this case, the task to be accomplished is more complex than the one faced by ROBY: the robot has to guide a visitor in the museum; whenever the visitor stops next to an exhibit, it has to wait and move again on the tour as the visitor starts moving again. Sometimes it may happen that the visitor moves away attracted by some interesting exhibit and in this case the robot has to follow her, re–plan the tour and start again the visit as soon as the visitor is ready.

The robot used in this indoor application has a differential drive kinematics with shaft encoders and it is provided with an omnidirectional vision system able to detect obstacles as well as landmarks in the environment. Not having a

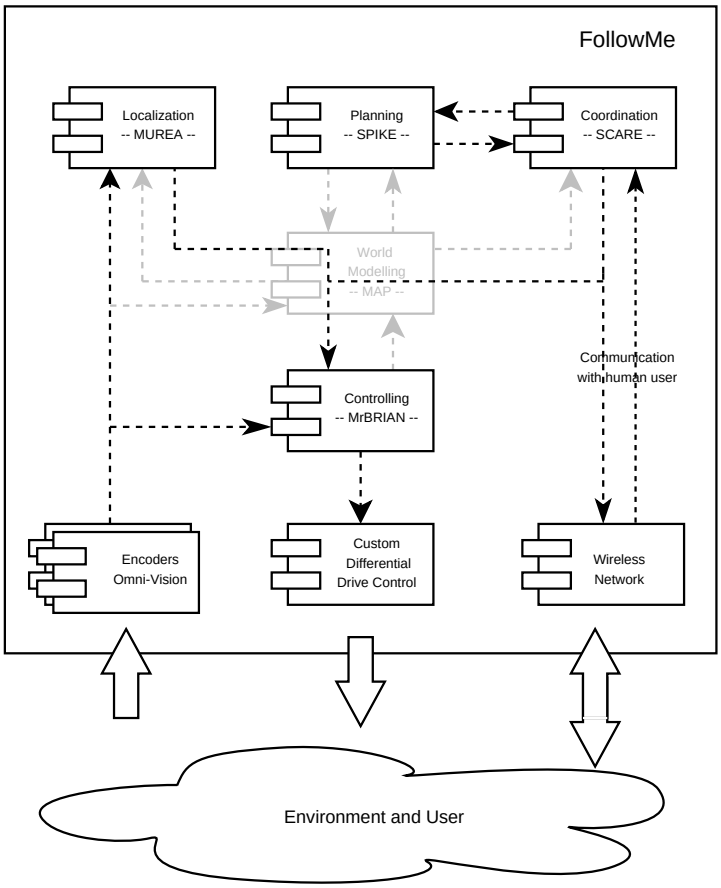

Fig. 5. Modules involved in the FollowMe architecture.

reliable positioning sensor like the Differential GPS, this application requires a more complex setup for the localization module that has to fuse sensor information and proposed actuation to get a reliable positioning. MUREA has thus a twofold role in this application: sensor fusion and self localization. Information coming from different sources is fused by using the framework of evidence, and the robot position is estimated as the pose that accumulate the most evidence [17].

Figure 5 describes the MRT modules involved in the FollowMe application. In this case the architecture includes also the trajectory planner SPIKE, triggered by the sequencing module SCARE whenever a new tour is required or a path has to be re–planned. Almost any behavior used in MrBRIAN and job in SCARE for the Roby application have been re–used, as the message containing points generated by SPIKE is equivalent to the mes-

sage transmitted by the external planner in that application. In addition to the forementioned `ReachPathPoint` and `AvoidObstacle` behaviors, here we have also the `SeekVisitor` and `WaitForVisitor` behaviors respectively in charge of getting close and leading the visitor. In the behavior hierarchy the latter behaviors subsume `ReachPathPoint` in order to properly perform the job and are subsumed by `AvoidObstacle` for obvious safety reasons. Notice that `AvoidObstacle` is exactly the same used in the ROBY application.

This time, the coordination is limited to the interface with the user to acquire the characteristics of the tour (i.e., destination, object of interests, and so on) and SCARE is mostly used as sequencer. Jobs as `SeekVisitorJob` and `ReachPathPointJob` have been implemented, but this time the latter was implemented as the combination of the `ReachPathPoint` and `WaitForVisitor` behaviors.

4.3 Milan RoboCup Team

We participate to the Middle Size League of the RoboCup competition [19] since 1998, at the beginning in the Italian National ART team, and since 2001 as Milan RoboCup Team. RoboCup provides a multi-robot domain with both cooperative and adversarial aspects, and it is perfectly suited to test the effectiveness of the Modular Robotics Toolkit in an environment different from the ones mentioned above.

The development of the Modular Robotics Toolkit followed a parallel evolution with the Robocup Team. At the very beginning there was only a reactive architecture implemented by BRIAN (i.e., the first version of MrBRIAN), after that, SCARE and MUREA followed, to realize an architecture resembling the ROBY case study presented before. While the planning module was developed to fulfill the needs of the FOLLOWME application, MAP has been developed to model the complex task of dealing with sensor fusion in a multi-robot scenario and opponent modelling.

The robots used in this indoor application have different kinematics; we adopt three differential drive custom platforms, two omnidirectional robots exploiting three omnidirectional wheels each, and one omnidirectional robot base with four omnidirectional wheels. Only differential drive platforms are provided with shaft encoders, while odometry for omnidirectional robots is computed using measures acquired by optical mice [5]. Omnidirectional vision is used to detect obstacles as well as the ball and landmarks in the environment.

The software architecture implemented with MRT to be used in these robots is quite different from the previous ones. Since RoboCup is a very dynamic environment the robot behaviors need to be mostly reactive and planning is not used. On the other hand, RoboCup is a multi–robot application and coordination is needed to exploit an effective team play. From the schema reported in Figure 6 it is possible to notice the central role that MAP, the world modelling module, plays in this scenario. Sensor measurements (i.e.,

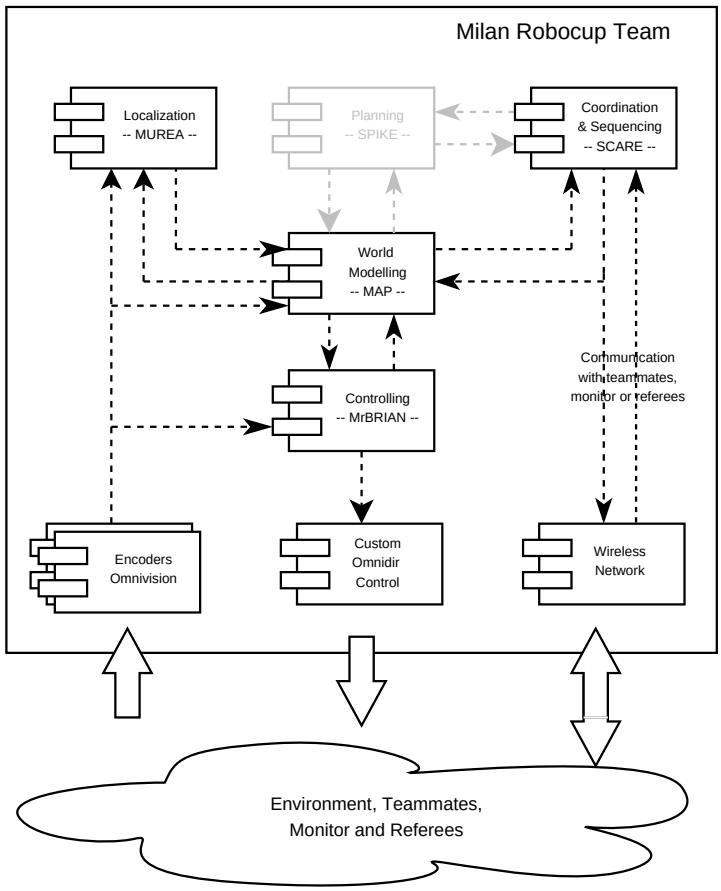

Fig. 6. Modules involved in the MRT architecture for the Milan RoboCup Team.

perceptions about visual landmarks and encoders when available) are fused with information coming from teammates to build a robust representation of the world. This sensor fusion provides a stable, enriched and up-to-date source of information for MrBRIAN and SCARE so that controlling and coordination can take advantage from perceptions and believes of other robots.

Coordination plays an important role in this application domain. We have implemented several jobs (`RecoverWanderingBallJob`, `BringBallJob`, `DefensJob`, etc.) and schemes involving few robots (`DefensiveDoubling`, `PassageToMate`, `Blocking`, etc.). These activities require the interaction of several behavioral modules. Also MrBRIAN has been fully exploited in this application; we have organized our behaviors, in a complex hierarchy, reported in Figure 7. At the first level we have put those behavioral modules whose activation is determined also by the information coming from the coordination

module. At this level, we find both purely reactive modules and parametric modules (i.e., modules that use coordination information introduced in MAP by SCARE). The higher levels contain only purely reactive behavioral modules, whose aim is to manage critical situations (e.g., avoiding collisions, or leaving the area after a timeout).

In order to give an idea of how the behavioral system works, let us consider the case of the `AvoidObstacle` behavior. The related behavioral module contains some rules which become active when it is detected any obstacle on the path followed by the robot, and they produce actions that modify the current trajectory in order to avoid collisions. Since avoiding obstacles is an important functionality, we have placed the related behavior at a high level of the hierarchy. The `AvoidObstacle` behavior is composed by a set of 32 rules similar to this one:

```
(AND (ObstacleNordNear)
     (ProposedTanSpeed FORWARD)) ⇒ (DEL.TanSpeed.*FORWARD)
                                   (TanSpeed STEADY)
```

that can be read as: "*if* I am facing any obstacle that is near *and* I would like to move forward *then* discard all the actions that propose a forward movement *and* propose to stay steady". Whenever the robot has no obstacle in its proximity, the `AvoidObstacle` module leaves all the proposed actions unchanged to the following level. Notice that the actions proposed by the behaviors at lower levels may be considered, but there is no need to know which behaviors have made the proposals. This interface among behaviors allows to implement independent behavioral modules that can be reused both in different hierarchies (e.g., we may exchange the levels of `AvoidObstacle` and of `KeepBallInKicker` if we would like a less aggressive behavior when the robot has the ball control) and in different applications (e.g., the same `AvoidObstacle` behavior has been used in all the three applications presented, although the subsumed behaviors are different)

5 Conclusions

In this paper, we have presented MRT, a modular software architecture that has been successfully adopted in several robotic applications. MRT consists of modules, each implementing one of the functionalities we have identified for the application. The modules interact with each other by exchanging XML messages supported by a dedicated middleware.

The modular implementation has made possible to reuse the functional modules and even the basic control modules implemented as behaviors, thus speeding up the development process. We have proved that with few weeks of work, and off–the–shelf modules integrated in a structured architecture such as MRT, it is possible to implement effective applications for autonomous robots.

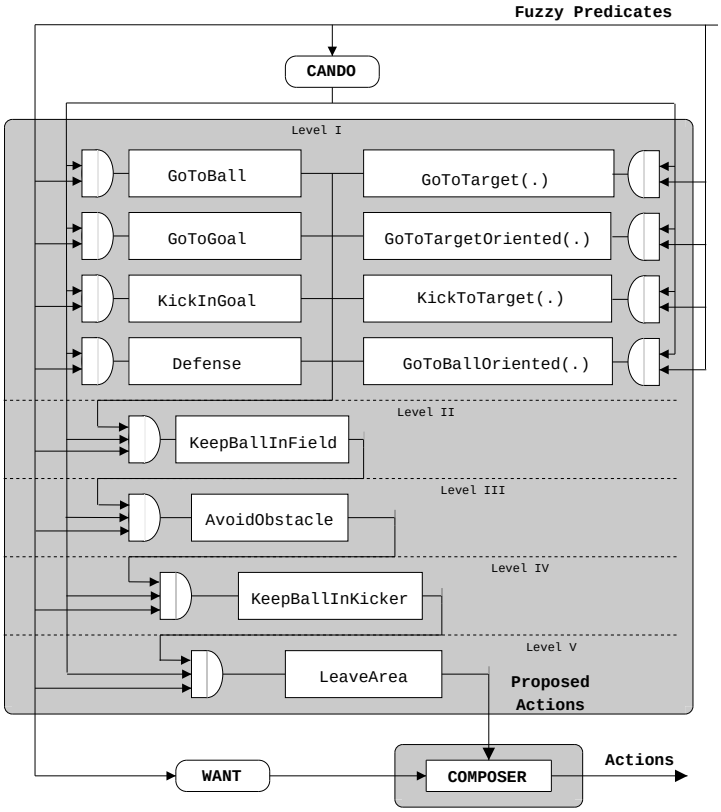

Fig. 7. The behavior architecture used by the Milan RoboCup Team.

Acknowledgements

This project has been partially supported by the PRIN 2002 Project MADSys (Software technologies and knowledge models for design, implementation and testing of multi-agent robotic system in real environments) co-funded by the Italian Ministry for Research, University and Technology (MURST).

References

1. Sven Behnke. Local multiresolution path planning. In Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 2004.
2. A. Bonarini, M. Matteucci, M. Restelli, and D. G. Sorrenti. Mrt, milan robocup team 2004. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*. Springer, 2005.

3. Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. Anchoring: do we need new solutions to an old problem or do we have old solutions for a new problem? In *Proceedings of the AAAI Fall Symposium on Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems*, pages 79–86, Menlo Park, CA, 2001. AAAI Press.
4. Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. A novel model to rule behavior interaction. In *Proceedings of IAS-8*, pages 199–206, 2004.
5. Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. Automatic error detection and reduction for an odometric sensor based on two optical mice. In *Proceedings of ICRA 2005*, pages 1687–1692. IEEE Press, 2005.
6. Andrea Bonarini and Marcello Restelli. An architecture to implement agents co-operating in dynamic environments. In *Proceedings of AAMAS 2002*, pages 1143–1144, 2002.
7. A. Brooks, T. Kaupp, A. Makarenko, A. Orebäck, and S. Williams. Towards component-based robotics. In *Proceedings of IROS 2005*, 2005.
8. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, April.
9. D. Brugali and M. E. Fayad. Distributed computing in robotics and automation. *IEEE Transactions on Robotics and Automation*, 18(4):409–420, 2002.
10. Riccardo Cassinis, Paolo Meriggi, Andrea Bonarini, and Matteo Matteucci. Device communities development toolkit: An introduction. In *Proceedings of EUROBOT 01*, 2001.
11. E. Gat. Three-layer architectures. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210. AAAI Press/The MIT Press, Menlo Park, CA, 1998.
12. C. D. Gill and W. D. Smart. Middleware for robots? In *Proceedings of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems*, 2002.
13. B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic. A domain-specific software architecture for adaptive intelligent systems. *IEEE Transactions on Software Engineering*, 21(4):288–301, 1995.
14. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995.
15. Matteo Matteucci. Publish/subscribe middleware for robotics: Requirements and state of the art. Technical Report 2003.3, Politecnico di Milano, Milan, Italy, 2003.
16. M. Piaggio, A. Sgorbissa, and R. Zaccaria. A programming environment for real-time control of distributed multiple robotic systems. *Advanced Robotic Journal*, 14(1):75–86, 2000.
17. Marcello Restelli, Domenico G. Sorrenti, and Fabio M. Marchese. Murea: a multi-resolution evidence accumulation method for robot localization in known environments. In *Proceedings of IROS 2002*, pages 415–420, 2002.
18. H. Utz, S. Sablatnog, S. Enderle, and G. K. Kraetzschmar. Miro - middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation. Special Issue on Object-Oriented Distributed Control Architectures*, 18:493–497, 2002.
19. Manuela M. Veloso, Tucker R. Balch, Peter Stone, Hiroaki Kitano, Fuminori Yamasaki, Ken Endo, Minoru Asada, Mansour Jamzad, B. S. Sadjad, Vahab S.

Mirrokni, Moslem Kazemi, Hmid Reza Chitsaz, A. Heydar Noori, Mohammad Taghi Hajiaghayi, and Ehsan Chiniforooshan. Robocup-2001: The fifth robotic soccer world championships. *AI Magazine*, 23(1):55–68, 2002.