

A Common Framework for Co-operative Robotics: an Open, Fault Tolerant Architecture for multi-league RoboCup Teams

Luís Mota^{1,2} and Luís Paulo Reis²

¹ Instituto Superior de Ciências do Trabalho e da Empresa (ISCTE), Portugal
luis.mota@iscte.pt

² NIADR - LIACC, Universidade do Porto, Portugal

Abstract. Research in the RoboCup domain has grown considerably since the beginning of this initiative more than ten years ago. Much of this growth is due to the existence of different leagues, that allow the focussing of research in specific and heterogeneous issues.

This specialisation of research has, though, proven to have some drawbacks: research subjects become very specific, and one loses the ability of properly generalising, and sharing, the obtained results.

This paper presents an architecture that aims at being open, enabling the development of independent components that can easily be ported between application environments. This architecture, called Common Framework, relies on standardised interfaces, protocols and communication channels between components. Besides allowing the free association of heterogeneous components, like real and simulated back-ends, it also considerably eases the introduction of principles of redundancy and fault tolerance.

1 Introduction

RoboCup is an international initiative to promote Artificial Intelligence, Robotics, and related fields. It fosters research by providing a standard problem where a wide range of technologies can be integrated and examined. RoboCup uses the soccer game as a central topic of research, aiming at innovations to be applied for socially significant problems and industries. Research topics include design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion.

The ultimate goal of the RoboCup initiative is "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team." This is certainly an ambitious goal, but research in co-operative robotics has been accumulating results that allow the community to continue believing in this challenge.

The RoboCup initiative has known how to attract research to a wide array of scientific problems, by creating different leagues that address specific and multiple questions. In the robotic soccer domain, in brief, there are simulation

leagues (2D and 3D) that abstract from all the hardware-related problems and allow focussing on higher-level problems; heterogeneous real robot leagues, in small and middle size, foster research in hardware development and player interaction and coordination; the standard platform league allows different teams to deploy different strategies in a standardised environment; and, finally, the humanoid league enables research on questions related to biped movement and playing skills. Furthermore, other initiatives such as the different Rescue and RoboCup@Home leagues extend the multi-robot research to domains in search and rescue and domestic scenarios, allowing the development of work in different, unrelated domains.

This diversity of application domains has certainly been responsible for attracting a wide and heterogeneous research community for this popular challenge, but these characteristics, associated with its strong competitive side, has also had some drawbacks. Namely, research teams, in order to stay competitive in the leagues they participate in, have normally focussed on a single league, trying to exploit its prevailing details. This specialization temptation has made it rare that teams simultaneously maintain competitive teams in different leagues, such as, e.g., simulation and middle size.

This natural specialisation tendency has brought noticeable drawbacks: teams achieve results that, though being competitive, are not easily generalisable and consequently shared between different leagues. E.g., it is not common to see high level results from the simulation leagues applied to middle size teams, where the same kind of challenges arise. This is certainly an undesirable result, which we try to deal with in this paper.

In section 2, a new robotic architecture that intends to be applicable to different leagues is presented. This architecture, which was named "Common Framework", relies on a multi-agent system (MAS) paradigm that is presented in section 3. The different components taking part in this MAS need to communicate using a language that is presented in section 4. Finally, some considerations about our proposed future work and conclusions are presented in sections 6 and 7.

2 Common Framework for Co-operative Robotics

This paper addresses the problem of developing a common approach to co-operative robotics with applications in domains where complex co-operative tasks must be performed by autonomous agents, like the different RoboCup competitions.

2.1 Requirements

The proposed architecture needs to address a set of requirements, in order to comply with its goals, as follows:

Open Architecture The architecture should be open, allowing the real-time addition and withdrawal of components without compromising its stability;

General application High-level components should be applicable to different leagues without further customisation;

Redundancy The architecture should allow the coexistence of redundant components, which may be co-ordinated, or selected, by other components.

2.2 Architecture layout

Since the architecture of the Common Framework is designed to be open and to include different components in real-time, these components must be able to communicate through a standardised interface, shared by all. The communication channels and protocols must also be common among all components.

The Common Framework includes a knowledge representation structure capable of representing organised information pertaining to the robotic soccer domain. In order to control different (simulated and real) robots, the Common Framework needs specific components that deal with each agent's perception and action capabilities. Low-level skills and perception mechanisms will be designed for each type of robot, while high-level actions can be chosen through the same, league-independent, decision-making component. A general action vocabulary will be developed to enable the low-level action components to understand high-level decision-making, whereas a perception vocabulary will address the representation of state-of-the-world information.

In order for the Common Framework to be truly flexible, allowing the integration and replacement of components in real time, it requires a flexible architecture that can be modified both in real and compile time. It is argued in the next section that the best way to answer these requirements is through a multi-agent system.

3 Multi-agent system architecture

In this paper, it is proposed to use a multi-agent system (MAS) for the control of each player. Thus, one team would be a system of multiple multi-agent systems. In each of the players, the same kind of components will exist (perception, action, decision, etc.), taking part on a MAS while using standardised communication. Each component might be implemented using a different programming language, or even be running in different machines, with distinct operating systems, as seen on Fig. 1. The components can arbitrarily vary in number, and even be redundant. This paper presents a proposal of the system to implement, making possible to exploit a scenario as depicted in this figure.

In order for the different components to interact freely, it is necessary that they have a way of knowing/discovering each other. With this purpose, there will be a communication management agent (ComMag) that will keep information on the existing components, as seen on section 3. Furthermore, this architecture requires a standardised communication language, for the expression of perception, action and state-of-the-world information. This language, described

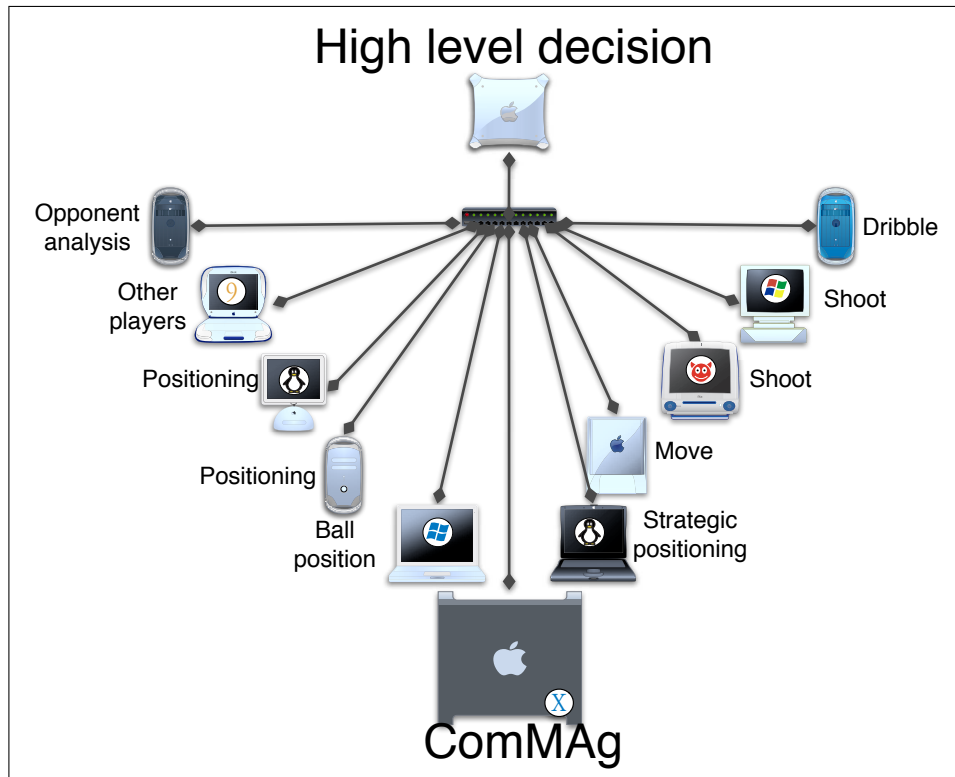


Fig. 1. Proposed architecture: arbitrary services, both in nature and quantity, can connect to the system.

in section 4, will include basic concepts, such as regions, locations and time, as well as soccer related items.

The communication management agent (ComMAg) was designed to keep information on all the existing agents, namely the addresses where these are accessible, and the type of service they provide. The agents will be accessible only through sockets, in order to keep the implementation simple and to allow maximum flexibility in the access: thus, these components could be implemented in any programming language, provided that they respect the communication interface. To achieve interoperability, the address for each agent must be known, as well as the port where it will be listening to connections, and the type of Service the agent provides. The necessary information is depicted in Fig. 2.

To precisely characterise each agent's services, a taxonomy of services that defines the concrete abilities of each agent must be created. Different agents will be able to supply more general or more specific services, e.g., an agent might be solely capable of executing a dribble, or of executing any type of physical action. Thus, the characterisation of the service might be done by a class in an upper

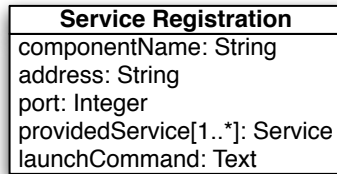


Fig. 2. Information needed for registering with the ComMAg.

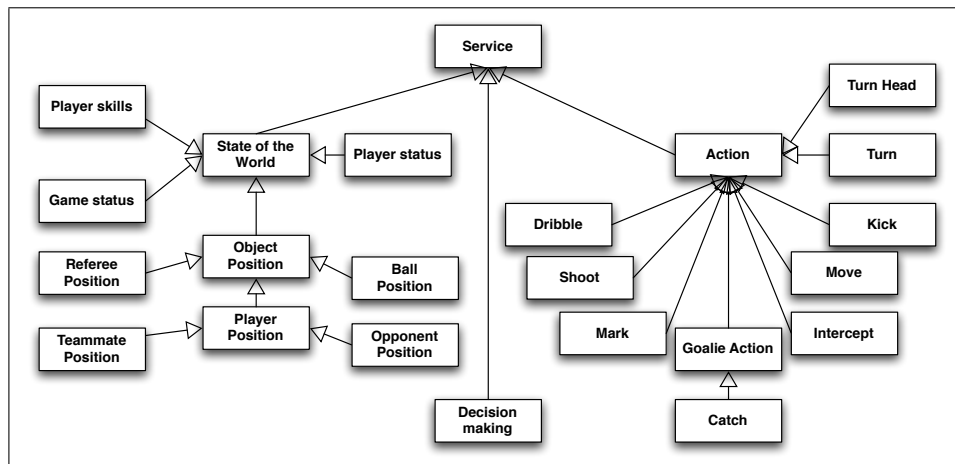


Fig. 3. Proposed hierarchy of soccer-related services.

level of the hierarchy (more general service), or further down (more specific). The proposed service hierarchy is summarised in Fig. 3.

The role of the ComMAg will be vital in the bootstrapping step, when the players are being launched: each component will register its services, and components that need other services to operate will look for the necessary components by querying the ComMAg. After this bootstrapping phase, the ComMAg could stay inactive, since the other components will be able to communicate directly among themselves. The ComMAg agent can, however, play another very important role, as argued in section 3.1: if, during the game, the ComMAg would keep track of the functioning of each of the components, it would be able to detect possible malfunctioning situations. Such an ability could be exploited in order to make the system fault tolerant.

3.1 Fault management

In this architecture, where components may be arbitrarily added to the system, and where redundant components are expected to exist, it is most appropriate to include a fault management mechanism.

To ensure that the components are behaving properly, the Common Framework expects each component to re-register with the ComMAg on a pre-defined schedule. If some component fails to do so after the elapsed time has passed, the ComMAg can assume that the component is malfunctioning, and react by trying to re-launch the agent using the registered *launchCommand*, included in the information sent by the agent for registration (see Fig. 2). If the component repeatedly fails to launch, it will simply be withdrawn from the registration directory. This component will cease to exist, and all interactions will have to be re-routed to other existing components.

4 Framework Language

The open and flexible architecture of the Common Framework demands that the various components are able to communicate according to a least commitment principle, since various components might be programmed in different languages, or be hosted in different machines. Thus, there has to be a standard interface to the components, and the exchange of messages also has to be pre-defined and standardised.

Furthermore, these concepts have to be integrated in a more general framework that allows their meaningful exchange between components. The components will need, e.g., to ask for the execution of actions, to demand the answering of queries, or to subscribe to important information sources. These communication primitives can be supplied by some agent communication language, like FIPA-ACL [1], which is an international standard, endorsed by IEEE, and supplies all the necessary primitives.

4.1 Language requirements

The most fundamental concepts are the ones pertaining to the description of the world and the possible actions of the players. These concepts should be modelled in an abstract way that should apply to different leagues and settings. This conceptualisation is visible in figures 4 and 5. The language underlying the Common Framework will have to address a set of requirements in order to be able to fulfil its role. These requirements will be described hereafter.

State of the world concepts The language needs to be able to express information about all the objects existing on the field, namely physical static objects, such as flags, the field and its regions, as well as dynamic objects such as the ball, the players and, possibly, a referee.

In the dynamic and fast changing domain of robotic soccer, the perception is always imperfect and error-prone. Therefore, the expression of locations must also be able to include information about the related uncertainty.

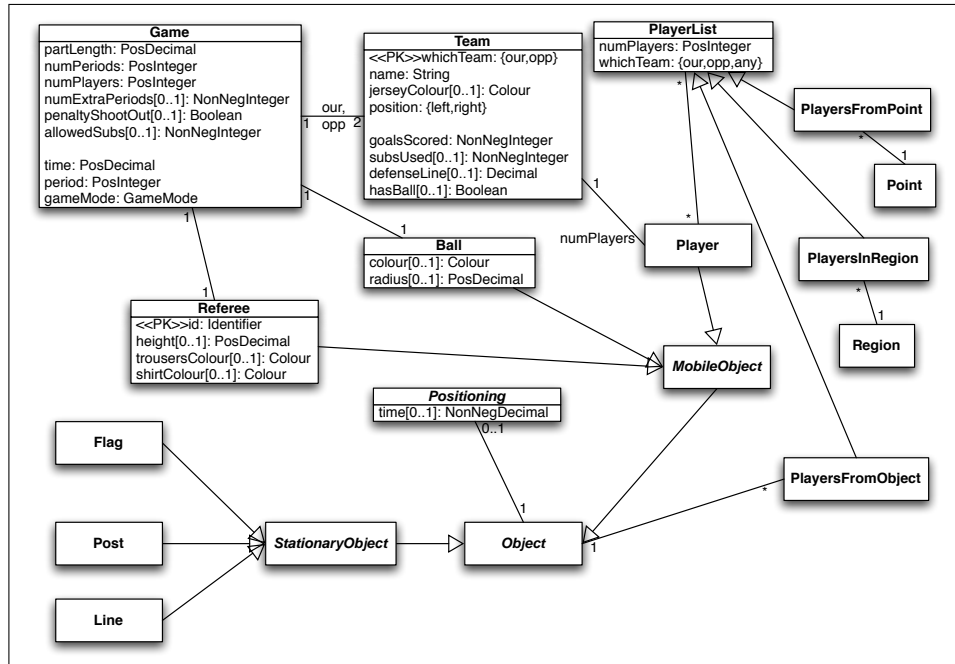


Fig. 4. General view of the framework's contents.

Robotic soccer actions There is obviously also the need for the expression of the actions that can be executed by the agents. Such actions should be modelled from a high level point of view, and include ball manipulation (kicks, shots), movement without ball (moving to a position, turn in a direction), as well as moving of the head, i.e., of the part of the robot where the usually existent cameras are located.

These requirements were considered closely and the resulting modelling can be seen in figure 4 (general view of the framework's contents), figure 5 (close view of the player's characterisation and description of the existing actions) and figure 6 (concepts related to positioning and associated uncertainties).

4.2 Language syntax

The model of the Common Frameworks needs its' concepts to be exchangeable through messages. Therefore, a syntax has to be defined that allows the different concepts in the model (instances, actions, access to attributes,...) to be expressed in a textual format.

The usage of well established agent systems' content languages, like FIPA-SL[2] or KIF[3] was considered, but was abandoned due to very different reasons:

- neither SL nor KIF allow the usage of Object Oriented concepts like methods;
- KIF has no support for actions;

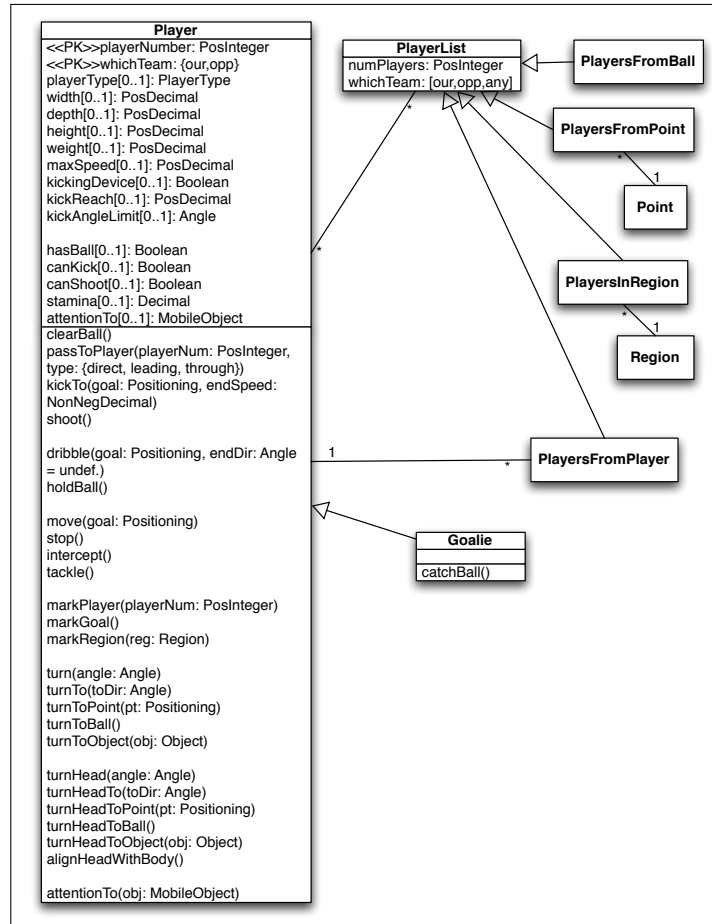


Fig. 5. View of the player’s characterisation and description of the existing actions.

– SL has very few available tools, namely a parser for C++.

Therefore, the proposed way to deal with OO-concepts is to create a simple, new approach to their expression is a textual format, which will now be presented.

Instances of classes The different classes in the model will need to have a way of expressing instances. With this purpose, it is suggested that the instances are expressed through S-expressions, with the class name as the first element and every attribute labelled by it’s name following a colon, e.g. ‘:name’. Using this formalism, an example of an Absolute Positioning would be as follows:

(AbsolutePositioning :x 10.3 :y -23.67 :yaw 0.23456)

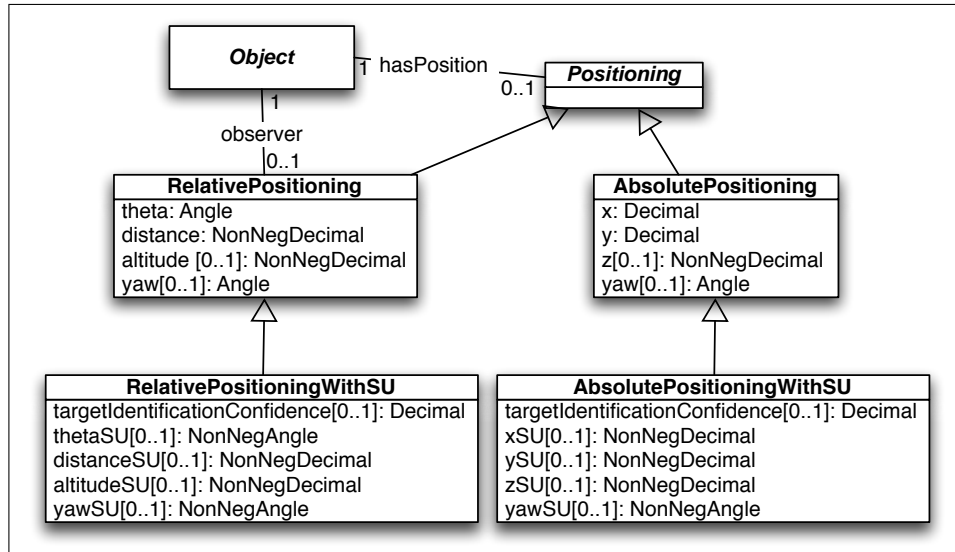


Fig. 6. Concepts related to positioning and associated uncertainties.

In terms of FIPA-ACL, these instances can be understood as propositions representing objects, and can be used, e.g., in the replies to queries.

Access to class attributes When querying parts of the state of the world, there will be the need to refer specific values of attributes, e.g., the positioning of a player. This need will be satisfied through a new primitive (val), which will be followed by the name of the sought class and attribute, and by the primary key to the class, when needed. As an example, the positioning of player five in the opponent team would include the player number and the team, since these are the primary key to this class:

```
(val Player positioning :playerNumber 5 :whichTeam opp)
```

To refer to the present time in the game, the expression would be even simpler, since there is only one game and therefore there is no need for the primary key:

```
(val Game time)
```

This attribute access mechanism corresponds to FIPA-ACL's referential expressions, and can be used as the content of a query.

Execution of methods The Player class defines methods modelling the possible actions of a player. These methods will need to be invoked by the high level decision component. With this purpose, the primitive *exec* is introduced. This primitive will need as parameters the name of the class, the primary key to the instance to which the method should apply, and the method's arguments. As an

example, in order for player 3 to turn to player 4, the following formulation is necessary:

```
(exec Player turnToObject :playerNum 3
:whichTeam our
:obj (Player :playerNum 4 :whichTeam our))
```

These method execution expressions can be considered action expressions, and therefore used as the content of FIPA-ACL requests.

5 Related work

Several authors have dealt with the concept of mobile robotic middleware in recent years. Some relevant approaches will be described in the current section.

Orocos [4, 5] intends to be a middleware for mobile robotics, strictly following free software best practices. It aims at being general purpose, and replace all proprietary drivers and control software shipped with the hardware. It also provides different software patterns to deal with common tasks, such as localisation determination. This framework has originated in an EU-funded project and has been developed for several years. The framework is in no way related to robotic soccer or the RoboCup initiative.

Miro [6, 7] is also a proposal for a middleware for mobile robotics, from a research team that has significant RoboCup experience[8]. The middleware, however, does not seem to have any specific adaptation for this domain. Similarly to Orocos, the framework is based on Corba³ principles. General purpose functionalities, such as mapping and localisation, are included in the framework.

Another middleware proposal[9] also originates from a team with RoboCup experience, but also pays no special attention to the domain, as it intends to be general-purpose and adaptable to different domains. The framework has three components, to deal with action and perception primitives, state-of-the-world, and high-level architectures, such as petri nets and state machines.

These different approaches intend to be of general application, with no specific support or adaptation for the RoboCup domain. From the point of view of a team working specifically on the robotic soccer domain, this generality brings disadvantages: the action and perception primitives are very basic, not covering essential actions like (different types of) kicks nor concepts like an off-side line, that become very difficult to deal with.

Moreover, these frameworks are normally extendable, therefore not providing a fixed set of primitives, nor a standardised soccer-specific vocabulary, as is the case with the Common Framework. This openness is undesirable, since it allows teams to develop league and team specific solutions and models, which will impair the sharing of results between teams and leagues.

Additionally, the reliance of Orocos and Miro on Corba shows that the frameworks are primarily intended to be used with Java and C++, since Corba offers little or no support for other languages like Prolog. The Common Framework

³ <http://www.corba.org>

relies simply on sockets, and can therefore be used with any programming language.

6 Future work

Having achieved a complete definition of the Common Framework and its underlying language, the next step will be to develop a pilot implementation in the 2D-simulation league. We intend to use public available implementations of well tested teams (e.g. UvA TriLearn [10] and Helios [11]) as our primary code source. At least two of these teams will be used, in order to show the interoperability of the architecture. Namely, it will be possible to simultaneously use code from different sources in the same agent.

Further on, it is intended to use the framework, with exactly the same controller agent, in the scope of the Mixed-Reality league [12], which uses actual, micro-sized robots.

As the framework becomes more stable and mature, it will also be used to control a team of robots in the Mid-size league [13], where new challenges linked to the usage of different, possibly failing sensors and actuators will arise, allowing the architecture to show the whole of its potential.

7 Conclusion

This paper has introduced a new architecture, a Common Framework for Cooperative Robotics, for the development of multi-robot teams. It aims at being open, flexible, redundant and fault tolerant. These qualities are inherent to the design of the Framework as a system of multiple multi-agent systems, where the enrolment of each player's components is managed by a centralised Communication Management Agent.

The existing components will communicate among themselves on basis of a shared modelling of the RoboCup domain, whereas the necessary conversations will need to comply to well-defined protocols that rule the development of meaningful interactions.

This open architecture allows the implementation of agents with arbitrary, redundant and reusable components. The components' relative independence also allows their separate development by diverse programming teams for posterior integration. Other advantages are the real-time addition and withdrawal of components, as well as tolerance to failures, and the usage of the same high-level agents in different environments (e.g. different leagues or real and simulated back-ends).

References

1. FIPA Technical Comitee: FIPA ACL message structure specification. Technical report, Foundation for Intelligent Physical Agents (2002)

2. FIPA Technical Comitee: FIPA SL content language specification. Technical report, Foundation for Intelligent Physical Agents (2002)
3. Genesereth, M.R., Fikes, R.E.: Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University (1992)
4. Bruyninckx, H.: Open robot control software: the orocos project. In: IEEE International Conference on Robotics and Automation. (2001)
5. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the orocos project. In: IEEE International Conference on Robotics and Automation. (2003)
6. Enderle, S., Utz, H., Sablatnög, S., Simon, S., Kraetzschmar, G., Palm, G.: Miro: Middleware for autonomous mobile robots. In: IFAC Conference on Telematics Applications in Automation and Robotics. (2001)
7. Utz, H., Sablatnög, S., Enderle, S., Kraetzschmar, G.: Miro—middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation* **18**(4) (8 2002) 493–497
8. Mayer, G., Kaufmann, U., Clauss, M., Hartmann, C., Monsch, M., Ruland, T., Seibold, B., Sitter, C., Wolf, F., Palm, G.: The ulm sparrows 2006 - team description paper. Technical report, University of Ulm (2006)
9. Ramos, N., Barbosa, M., Lima, P.: Multi-robot systems middleware applied to soccer robots. In: ROBOTICA 2007 - 7th Portuguese Robotics Festival. (2007)
10. Kok, J.R., Vlassis, N., Groen, F.: Uva trilearn 2004 team description. Technical report, University of Amsterdam (2004)
11. : rctools Web Page, <http://rctools.sourceforge.jp/akiyama/>
12. Gimenes, R., Mota, L., Reis, L.P., Lau, N., Certo, J.: Simulation meets reality: A cooperative approach to robocup's physical visualization soccer league. In: 13th Portuguese Conference on Artificial Intelligence, EPIA 2007. (2007)
13. Moreira, A.P., Costa, P., Scolari, A., Sousa, A., Marques, P.: 5dpo-2000 team description for robocup 2006. Technical report, University of Porto (UP) (2006)