# Flexible Synchronisation within RoboCup Environment: a Comparative Analysis

Marc Butler, Mikhail Prokopenko, Thomas Howard

Artificial Intelligence in e-Business Group
CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, Australia

**Abstract.** Synchronisation between an agent and the environment it resides in, is without a doubt, an important aspect of a more generic problem of agent interaction with the environment. A systematic comparative analysis of alternative approaches to the synchronisation problem remains an open challenge, despite numerous successful implementations of RoboCup teams in the past. The underlying reasons appear to be a multiplicity of software platforms, implementation changes in the Simulator itself, and sometimes a methodological bias of designers driven by a particular agent architecture. In this paper we describe alternative methods of agent-environment synchronisation, introduce a simple software tool for analysing RoboCup games via server log files, and compare the proposed synchronisation alternatives with respect to certain quantitative metrics. This comparative analysis is conducted without varying situated, tactical or strategic agent skills, highlighting purely synchronisation features.

## 1 Introduction

A distinction between a softbot (synthetic agent) and a software program (module or subroutine) has been extensively studied in context of multi-agent systems. Agent characteristics such as self-containment, temporal continuity, reactivity, pro-activeness, autonomy, etc. are often used to illustrate properties that are unique to agency, as opposed to software modules. In general, agents are supposed to make their decisions and update their behaviour on the basis of local, rather than global, information. Multi-agent interactions lead to emergent patterns in overall system behaviour. In general, emergent behaviour cannot be predicted or even envisioned from knowledge of what each component does in isolation [1]. We believe that one of the most significant aspects making an agent something more than just a software module, is its existence in an environment, its interactions with the environment, and ultimately its adaptation to the environment.

The Simulation League of the RoboCup provides a standard competition platform where teams of software agents play against each other [2]. As in the real game, "players" have only fragmented, localised and imprecise information of the field, and must respond to actions and events in limited time. Recent

RoboCup literature investigated various aspects of RoboCup simulation, including learning of basic and tactical skills [8], genetic evolution of agent behaviors [3], different levels of agents reasoning abilities [6], cooperation between agents [9], opponent modelling [10], etc. The Simulator which creates the environment has also been comprehensively described on both implementation and semantic levels [2, 5]. A particular problem of agent's synchronisation with the environment has been addressed as well, for example in [4]. However, we believe that a systematic comparative analysis of different approaches to the synchronisation problem remains an open challenge. We feel that such analysis could shed more light on the nature of agents interactions with the RoboCup environment.

In this paper we describe alternative methods of agent-environment synchronisation, introduce a simple software tool for analysing RoboCup games via log files, and compare the proposed synchronisation alternatives with respect to certain quantitative metrics. It is worth pointing out that this comparative analysis is conducted without varying situated, tactical or strategic agent skills, highlighting purely synchronisation features.

## 2 Synchronisation Alternatives

The RoboCup Simulation League provides essentially a pseudo real-time environment. This platform exposes the inherently problematic nature of timing which, as a consequence, results in many engineering problems. Client host platforms typically vary greatly in the available time resolution at both the hardware and in system library level. In addition there is the issue of network speed and reliability, and available resources such as CPU time and memory.

One of the implementation challenges posed by this is synchronisation with the Soccer Server, which we will refer to as the "server". By default the server simulates time periods of 100ms known as "server cycles", in which a client may send a command. Contiguous to this, sense_body information (kinematic parameters, stamina, etc) is sent at 100ms intervals, and visual information at 150ms intervals. As documented, the server will execute one and only one command per cycle received from any single client, hence the challenge is to ensure that at most one command is sent. We call the detrimental situation where a client sends two or more commands in a server cycle a "clash". Another potentially harmful situation is where no commands are sent in a server cycle, even though commands are sent in each adjacent server cycle; we call this a "hole". For example we would expect a client chasing the ball to have an unbroken sequence of dash and turn commands until it reaches the ball. Occurrences of holes generally slow the agents, and hence disadvantage them in intercepting and running for the ball.

We consider only the default situation ignoring the complexities introduced by the alternative visual information delivery periods provided at the cost of either cone of vision, quality or both. Most obviously we are faced with the problem that 50 % of all visual information is delivered mid server cycle, and every third cycle a client may receive no current information at all. Secondly, we

also find that there is some elasticity in the server cycles. That is, the windows may not always be 100ms, they are generally never less 100ms, but according to the resources available and calculations required they may grow, by multiples of 10ms. From an implementation perspective, inaccuracies in the system timing routines are due to timer resolution, limited resources, and the process-scheduling algorithm employed. There is some latency (though it may be negligible), in the distributed multi-process environment in which the simulation takes place. Lastly, packets may be irrevocably lost in UDP transmission; fortunately this is very uncommon. In summary, synchronisation with "elastic" server cycles in the presence of asynchronous visual information becomes a challenging task.

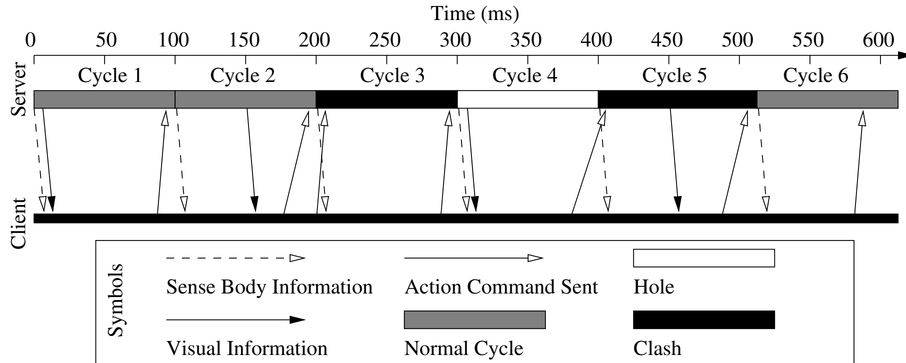


**Fig. 1.** Synchronisation problem.

Our clients are multi-threaded, where a single thread is dedicated to sensing (watching the UDP port) and reasoning, and another is for acting (dispatching commands, and subsequently synchronisation). Upon receiving visual information the client determines which type of commands and in what sequence they should be executed. The number of commands inserted typically falls within the range of 0 to 4. These are then placed in a thread-safe queue. The commands are extracted one at a time from the queue as determined by the acting thread. We have found many possible synchronisation variations, and have distilled these into four categories, which seem to address the dominant aspects of the problem.

These scheduling schemes can be conveniently categorized according to certain traits. Firstly we say the timing mechanism is either, external or internal. External timing can be thought of as observation of change in the environment, which is expressed conveniently as the sense_body information in this domain. Internal timing can be thought of as a biological clock or regulation mechanism, realized by our use of the operating system timing routines. Separate to the timing mechanism, we may further classify a synchronisation schema as "On Demand". "On Demand", may be thought of as "as immediately needed" or "as immediately required".

### 2.1 Internal Basic

The internal basic scheme is the simplest, and appears to provide the least optimal performance. The acting thread counts out intervals of 100ms, and then attempts to send a command to the server if one is available. This works on the premise that if commands are sent only at 100ms intervals, these adjacent time points will occur within adjacent server cycles. This methodology does not directly address the issue of mid cycle information or cycle elasticity. It does indirectly, in a probabilistic way, address the issue as the time point may occur anywhere within the window. However, this illuminates an additional problem. Even given the situation of inelastic cycles, commands dispatched very close to the end of a server cycle (typically within the last 10ms) may arrive in either the current cycle or next in stochastic fashion; somewhat like an erratic pendulum (see, for example, cycle 4 in Figure 2). In short, the internal basic scheme may work only under certain assumptions, achieving good synchronisation on probabilistic average. However successful expression of some behaviors may demand more precise synchronisation.

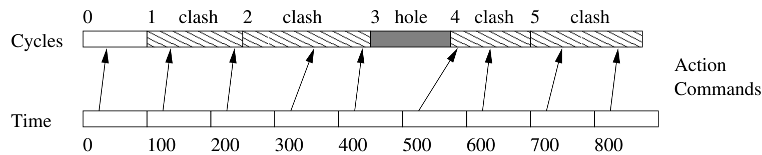


**Fig. 2.** Synchronisation with internal basic scheme.

### 2.2 Internal On-Demand

This scheme varies from the Basic implementation in two ways. Firstly, in addition to counting the 100ms intervals, the acting thread now receives notice of the insertion of commands into the queue. Secondly, if no command was sent at the last interval, and there is no command to be sent at the current interval, the thread will sleep until a command is inserted in the queue. This may advantage the agents by providing them with immediate response to sudden events. However, the internal on-demand scheme also suffers from an inability to "guess" a server cycle's start point and duration. Neither of the internal schemes adequately addresses the issue of mid cycle information or cycle elasticity and both appear to be greatly limited in utility.

### 2.3 External Basic

In this schema the sense_body information is utilized to indicate the commencement of a new server cycle. This information allows us to use the concept of a "window". This is a time period that indicates the "window of opportunity" in which the client may reason and subsequently place commands in the queue during which time the acting thread will not attempt to dispatch the first command from the queue. This directly addresses the issue of elasticity in server cycles, as the sense_body information is also subject to this condition (i.e., if server cycle expands, the sense_body interval follows). The issue of mid-cycle information

may also be addressed by selecting a window size of 0.5 of a server cycle plus the average time needed to generate new commands (as during cycle 5 in Figure 3). Obviously, if the time taken exceeds the average, the mid-cycle information will not be utilized (see, for example, cycle 2 in Figure 3).

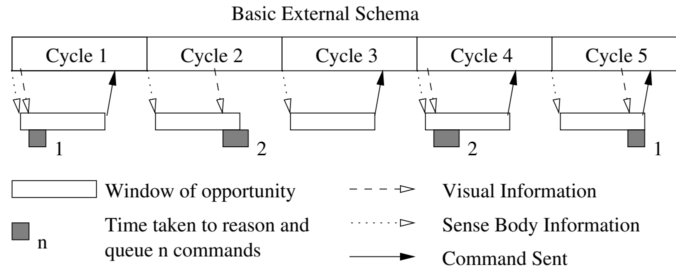


**Fig. 3.** Synchronisation with external basic scheme.

### 2.4 Flexible Windowing

In order to attempt to realize the optimal situation as often as possible, Flexible Windowing attempts to fuse the concept of "On-Demand" with that of "Windowing". This schema is almost identical to External Basic Windowing with the exception, that the amount of time before dispatching a command is varied according to the state of the queue. If the queue was emptied in the previous window, the action thread may select to sleep for a different (typically shorter) duration, than it would if the queue still contains commands — see, for instance, cycle 2 in Figure 4, where the mid-cycle information is utilized and a command is dispatched at the end of a (possibly) shorter window. A special case is where the time to wait in both circumstances is equal, and we call this "Fixed Windowing". In this case the state of the queue is irrelevant.

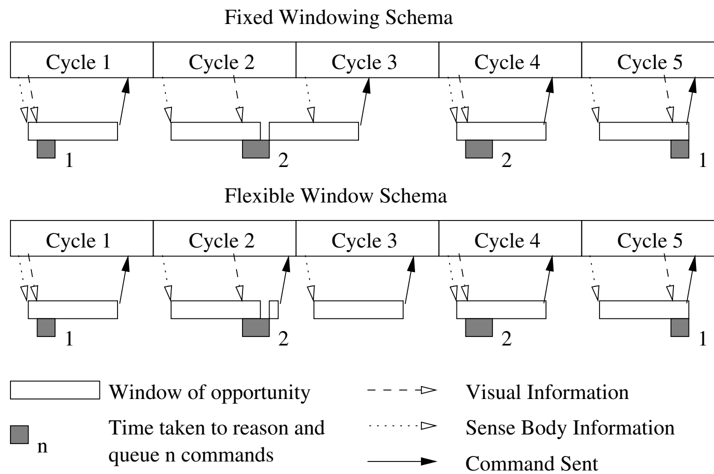


**Fig. 4.** Flexible synchronisation.

## 3 Analysis of Game Logs

The AGL (Analysis of Game Log) tool was initially developed to provide an empirical measurement of the success of our alternative synchronisation schemes. This is made possible by an option in the Soccer Server configuration file, which enables the server to maintain a log of the commands sent by each client. We will refer to this file as the "Game Log". The game log layout is simple to parse, yet contains considerable information. Each line consists of the id of the server cycle, the player id and the command sent verbatim. The following is an extract from a game log, selecting commands received from a particular client, in this case number 3 of the "Fixed Windowing" team:

2080 $Recv$ $fixedwin_3$ : ($turn$ $-29.76$)
2082 $Recv$ $fixedwin_3$ : ($change_view$ $normal$ $high$)
2082 $Recv$ $fixedwin_3$ : ($turn$ 17.52)
2083 $Recv$ $fixedwin_3$ : ($dash$ 35.00)
2085 $Recv$ $fixedwin_3$ : ($dash$ 35.00)
2086 $Recv$ $fixedwin_3$ : ($dash$ 35.00)
2089 $Recv$ $fixedwin_3$ : ($dash$ 100.00)
2089 $Recv$ $fixedwin_3$ : ($dash$ 100.00)
2091 $Recv$ $fixedwin_3$ : ($dash$ 100.00)

It is easy to see, for example, a "clash" occurring at server cycle 2089, followed by a potential "hole" at 2090.

Currently the AGL is implemented as a Perl 5 script, which not only parses and collates data from the log file, but also provides a terse analysis. The AGL tool is sufficiently flexible to be able to handle partial games. Many (though not all) measurements may be made with any variable number of players from 1 to a full 11. In addition to the obvious comparison of goals scored, the AGL calculates a number of measurements, provided both as a count (raw data), and as a percentage. The meaning of the percentage varies between measurements and is described separately with each formula.

It is well-documented that some of the client commands can be executed in parallel during one server cycle (for example, "say" or "change view"). Others, like "dash", "turn", "kick" and "catch", are executed only once per cycle. We are mostly interested in synchronising the latter command type, which we shall refer to as "action" commands, or "actions". Let us introduce the following notation:

$\alpha_{i,j}$ is a function returning the number of all action commands received by the server from a player $i$ at cycle $j$;
$\lambda_{i,j}$ is a function returning 1 if the server received one or more action commands from a player $i$ at cycle $j$, and 0 otherwise;
$\delta_{i,j}$ is a boolean function returning $true$ if $\alpha_{i,j} = 0$, and $false$ otherwise;
$\gamma_{i,j}$ is a boolean function returning $true$ if the server received a "dash" action command from a player $i$ at cycle $j$, and $false$ otherwise;
$\theta(a)$ is a function returning 1 if a boolean expression $a$ is true, and 0 otherwise.

In short, $\lambda_{i,j}$ is 1 if the cycle $j$ was used by the player $i$ (even if more than once), while $\delta_{i,j}$ represents unused cycles.

"Activity" measurement is given by the formula:

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{i,j}}{n*m}$$

where $n$ is a number of players, and $m$ is a number of server cycles.

An activity level of 100 % indicates that each client in the team sent an action command in every cycle. Activity provides some interesting information. In particular, we have observed that a winning side often has a lesser activity, possibly making a losing side chase the ball while denying them a good passing game.

"Clashes ratio" measurement is given by the formula:

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(\alpha_{i,j}-\lambda_{i,j})}{\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{i,j}} = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{i,j}}$$

where $n$ is a number of players, and $m$ is a number of server cycles.

Basically, this ratio represents the frequency of clashes with respect to all action commands, and is primarily used to determine the quality of synchronisation between the clients and the server.

"Holes ratio" measurement is given by the formula:

$$\frac{\sum_{i=1}^{n}\sum_{j=2}^{m-1}\theta(\gamma_{i,j-1}\wedge\delta_{i,j}\wedge\gamma_{i,j+1})}{\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{i,j}}$$

where $n$ is a number of players, and $m$ is a number of server cycles.

Intuitively, this ratio represents the sum of all unused cycles $j$ occurring between the adjacent cycles $j-1$ and $j+1$ in which the client sent "dash" commands, calculated for the whole team in proportion to the total number of action commands sent (including clashes) for players on the same team. Holes (more precisely, dash holes) are used in conjunction with clashes to determine the quality of synchronisation between the clients and the server.

Here is an example report produced by the AGL for two teams (after a 6000 cycles game):

| Team: flexwin | | | Team: fixedwin | | |
|---|---|---|---|---|---|
| All commands.. | 48276 | (73.15%) | All commands.. | 46414 | (70.32%) |
| Activity............ | 41443 | (62.79%) | Activity............ | 39988 | (60.59%) |
| Holes................... | 164 | (0.34%) | Holes................... | 259 | (0.56%) |
| Clashes.............. | 2544 | (5.27%) | Clashes.............. | 1340 | (2.89%) |
| Kicks................... | 479 | (0.99%) | Kicks................... | 462 | (1.00%) |
| Dashes............. | 22785 | (47.20%) | Dashes............. | 21323 | (45.94%) |
| Turns............... | 18079 | (37.45%) | Turns............... | 18181 | (39.17%) |
| Catches............... | 100 | (0.21%) | Catches................. | 22 | (0.05%) |

The simple format of the game log requires low resource overhead to generate during runtime, thus minimising impact on the experiment. The time required by AGL to generate a report such as the one above is quite small, though proportional to the size of the log. This provides information promptly after the completion of the game, allowing an experimenter to quickly confirm their impressions. The log also contains an identical depth of information for both teams participating in the game. This enables an analysis of games involving heterogeneous teams, provided they use the same version of the Soccer Server protocol.

## 4 Preliminary Comparative Analysis

In this section we briefly describe an experiment which exemplifies how the AGL tool is used in comparative analysis of alternative synchronisation schemes summarised in section 2. Four schemes were selected for the experiment: internal on demand, basic external, flexible windowing and fixed windowing. Consequently, four different teams were compiled, and played 15 round-robin tournaments (first 3 tournaments with an aggressive tactical formation 3-5-2, and the next 12 tournaments with a more conservative 5-3-2 formation), resulting in 45 games played by each team (9 in the first experiment and 36 in the second). The overall competition performance is summarised in the following tables (a win earns 3 points, and a draw 1 point).

| | Internal | Basic External | Flex Windowing | Fixed Windowing |
|---|---|---|---|---|
| Wins | 0 | 5 | 7 | 4 |
| Losses | 8 | 3 | 1 | 4 |
| Draws | 1 | 1 | 1 | 1 |
| Goals For | 14 | 36 | 42 | 30 |
| Goals Against | 34 | 23 | 20 | 25 |
| Total Points | 1 | 16 | 22 | 13 |

**Table 1.** Results with the 3-5-2 formation.

| | Internal | Basic External | Flex Windowing | Fixed Windowing |
|---|---|---|---|---|
| Wins | 8 | 16 | 16 | 9 |
| Losses | 18 | 8 | 8 | 15 |
| Draws | 10 | 12 | 12 | 12 |
| Goals For | 22 | 37 | 39 | 26 |
| Goals Against | 36 | 26 | 30 | 32 |
| Total Points | 34 | 60 | 60 | 39 |

**Table 2.** Results with the 5-3-2 formation.

These results clearly rule out the "internal on demand" synchronisation — the corresponding team managed to get only one draw in 9 games, losing 8 times, with the 3-5-2 formation, and finished clear last with the 5-3-2 formation.

Three other teams were quite competitive — no one escaped without losing to some other team in the first short experiment, and the next experiment has shown some close scores as well. At the end, "flexible windowing" edged ahead overall, and "fixed windowing" finished clear third. "Basic external" proved to be a solid performer. In fact, it shared the first place with "flexible windowing" in the second experiment. It should be pointed out, however, that more games with alternative formations should be played to get more statistically significant results, and other placements of these three external-based schemes are possible.

At this stage, we intended just to demonstrate how tournament results can be complemented by the AGL tool. The following table contains average and standard deviation measurements for "activity", "clash" and "dash holes" ratios.

| *Name* | Clashes | | Holes | | Activity | |
|---|---|---|---|---|---|---|
| | Average | Std. Dev | Average | Std. Dev | Average | Std. Dev |
| Internal | 3.67 | 0.70 | 0.21 | 0.03 | 68.05 | 3.61 |
| Basic External | 3.01 | 2.14 | 0.57 | 0.11 | 58.49 | 4.45 |
| Flexible Windowing | 5.26 | 0.73 | 0.46 | 0.08 | 61.12 | 2.59 |
| Fixed Windowing | 3.01 | 1.82 | 0.59 | 0.10 | 58.92 | 3.47 |

**Table 3.** AGL statistics with the 3-5-2 formation.

| *Name* | Clashes | | Holes | | Activity | |
|---|---|---|---|---|---|---|
| | Average | Std. Dev | Average | Std. Dev | Average | Std. Dev |
| Internal | 0.31 | 0.22 | 0.16 | 0.03 | 61.45 | 3.67 |
| Basic External | 0.03 | 0.03 | 0.64 | 0.09 | 52.70 | 3.47 |
| Flexible Windowing | 1.19 | 0.44 | 0.50 | 0.09 | 55.00 | 4.02 |
| Fixed Windowing | 0.08 | 0.06 | 0.60 | 0.10 | 53.81 | 3.59 |

**Table 4.** AGL statistics with the 5-3-2 formation.

It should be noted that the second experiment was conducted in a better networking environment. It appears that the tournament winner has, in fact, a higher percentage in the "clash ratio" — but with a significantly lesser standard deviation in the first experiment, making this scheme more stable and robust under less "friendly" conditions. In addition, its "holes ratio" is smaller among three best teams in both experiments. The "internal" scheme exhibited quite comparable ratios, but the team was too much active — reflecting the fact that a large portion of commands arrived at incorrect or non-optimal cycle, and the players had to chase the ball more than opponents. Again, more games would definitely provide statistically better results.

## 5 Conclusions

We felt that mechanical analysis of game logs added greatly to comparison of goals scored, and observations by what are now "expert eyes" amongst the devel-

opment team. In developing AGL, we have continued to find that more involved analysis of the log may provide us with a multitude of parameters by which we can compare teams from various stages of development with one another, as well as other teams.

Our preliminary comparative analysis was carried out without varying situated, tactical or strategic agent skills and highlighted purely synchronisation issues. The agents used in the experiments were developed in accordance with the Deep Behaviour Projection agent architecture, which provided a systematic support for design and full implementation of the team Cyberoos [7]. In general, agent-environment synchronisation is only one factor contributing to a teams overall performance. The precise nature of relationships and dependencies between synchronisation schemes and the agent architecture is a subject of future research.

## References


1. John L. Casti. Would-be Worlds. How Simulation is Changing the Frontiers of Science. John Wiley & Sons, 1997.
2. Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela M. Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda and Minoru Asada. The RoboCup Synthetic Agent Challenge. In Proceedings of the 15th International Joint Conference on Artificial Intelligence, 1997.
3. Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson and James Hendler. Co-evolving Soccer Softbot Team Coordination with Genetic Programming In RoboCup-97: Robot Soccer World Cup I (Lecture Notes in Artificial Intelligence No. 1395), H. Kitano, ed. Berlin: Springer-Verlag, 398–411, 1998.
4. Kostas Kostiadis and Huosheng Hu. A Multi-threaded Approach to Simulated Soccer Agents for the RoboCup Competition, In Proceedings of the IJCAI'99 RoboCup Workshop, Stockholm, August 1999.
5. Mikhail Prokopenko, Ryszard Kowalczyk, Maria Lee and Wai-Yat Wong. Designing and Modelling Situated Agents Systematically: Cyberoos'98. In Proceedings of the PRICAI-98 Workshop on RoboCup, 75–89. Singapore 1998.
6. Mikhail Prokopenko and Marc Butler. Tactical Reasoning in Synthetic Multi-Agent Systems: a Case Study. In Proceedings of the IJCAI-99 Workshop on Nonmonotonic Reasoning, Action and Change, 57–64. Stockholm 1999.
7. Mikhail Prokopenko, Marc Butler and Thomas Howard. On Emergence of Scalable Tactical and Strategic Behaviour. To appear in Proceedings of the RoboCup-2000 Workshop, Melbourne 2000.
8. Peter Stone and Manuela Veloso. Team-Partitioned, Opaque-Transition Reinforcement Learning. In Proceedings of the Third International Conference on Autonomous Agents (Agents'99) and in "RoboCup-98: Robot Soccer World Cup II", M. Asada and H. Kitano (eds.), Springer Verlag, Berlin, 1999.
9. Peter Stone and Manuela Veloso. Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork. In Artificial Intelligence, volume 100, number 2, June 1999.
10. Peter Stone, Patrick Riley, and Manuela Veloso. Defining and Using Ideal Teammate and Opponent Agent Models. In Proceedings of the Twelfth Innovative Applications of AI Conference (IAAI-2000).