PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 10/06

A Multi-agent Framework to Retrieve and Publish Information on Qualification and Elimination Data in Sports Tournaments

Thiago Ferreira de Noronha Carlos José Pereira de Lucena
Celso Carneiro Ribeiro Sebastián Urrutia

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 10/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena March, 2006

A Multi-agent Framework to Retrieve and Publish Information on Qualification and Elimination Data in Sports Tournaments

Thiago Ferreira de Noronha, Carlos José Pereira de Lucena, Celso Carneiro Ribeiro and Sebastián Urrutia

{tfn,lucena,celso,useba}@inf.puc-rio.br

Abstract. This work proposes an object-oriented framework for implementing a multi-agent system to collect, process, and publish information in the web. The primary objective is to create a framework for developing autonomous applications related to qualification and elimination problems in sports tournaments. These applications involve collecting results from several sources, processing them, and publishing a report on the situation of each team taking part in the competition, regarding qualification and elimination statistics. An instance of this framework was created to follow the Brazilian national soccer tournament and other soccer competitions.

Keywords: Agents, Framework, Futmax, Sports, Integer programming.

Resumo. Este trabalho propõe um framework orientado a objetos para implementar um sistema multi-agentes responsável por realizar operações de coleta, processamento e publicação de informações na WEB. O objetivo principal é criar uma infra-estrutura para desenvolver aplicações autônomas relacionadas com problemas de classificação em campeonatos esportivos. Estas aplicações caracterizam-se por coletar os resultados dos jogos de uma fonte, processá-los e publicar um relatório sobre a situação de cada time dentro da competição. Uma instância deste framework foi criada para acompanhar o campeonato brasileiro e outros torneios de futebol.

Palavras-chave: Agentes, Framework, Futmax, Esportes, Programação inteira.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

Sports tournaments are followed by millions of people across the world, continually eager for information on the situation of their teams in each competition. Fans check newspapers, radio, television and, more recently, the Internet daily for information on the performance and qualification chances of their favorite teams. Most of the information supplied through these communication channels is statistical data, which are calculated using each team's history and some "magic" numbers defined through the observation of previous tournaments.

A number of papers using combinatorial optimization techniques have been published with the aim of providing more precise information on the situation of each team within a tournament, see e.g. [14, 4, 26, 25, 19, 1, 28, 22, 23, 8]. They enable users to calculate when a team is mathematically the champion of the competition, when it is mathematically qualified for the next phase of the competition, whether it depends only on its own results to qualify, whether the team depends on other results to qualify, whether the team no longer has any chance of qualifying, and so on.

This article presents a framework [11] to help in the development of such applications, which can usually be divided into three stages: (1) collection of game results from one or more sources, (2) solution of one or more combinatorial optimization problems, and (3) publication of information on each team via the Internet.

The framework was modeled using the multi-agent systems software engineering paradigm [17, 29, 13, 27] and implemented in Java. An instance of the framework was generated to automate the management and the maintenance of the website of the FutMax project [24], that tracks the Brazilian soccer tournament and other soccer competitions.

The FutMax project and its website were created in 2002. Update and publication procedures in the first two years of operation were fairly different from the system proposed in this work and currently used. Collecting, processing, and publishing stages were executed separately by independent programs. To collect game results, a human operator checked periodically the official web page of the Brazilian tournament and edited a text file with the results. Next, the same operator was in charge of running the program responsible for obtaining qualification and elimination statistics, feeding in the game results file as input and receiving as the output another file containing qualification and elimination statistics for each team. The output file was then emailed to a web designer, who was supposed to update the FutMax site with the qualification and elimination statistics.

This process is almost identical for many applications related to qualification and elimination prediction in sports tournaments, irrespective of the sport in question and the tournament rules. This fact prompted the development of a framework to provide assistance to such applications. An overview of the methodology employed in this work is described in the next section. Section 3 presents in detail the instantiation of this framework to the Brazilian soccer tournament. In Section 4, this work is compared with similar works reported in the literature. Finally, a number of conclusions and potential extensions are presented in Section 5.

2 Methodology

The methodology employed in this work is divided into two parts. First, a model with reactive software agents is proposed. It is followed by the design of an agent-oriented framework for implementing data collection, processing, and publishing applications related to qualification and elimination problems in sports tournaments.

2.1 Agent modeling

Agents are a general purpose software abstraction. They are the next significant software abstraction, especially for distributed systems. A software agent is a piece of autonomous or semi-autonomous, proactive, and reactive computer software. The multi-agent paradigm uses agents that cooperate independently by exchange of messages in order to solve a general problem. A system of such agents is dynamic and flexible in varying conditions and furthermore provides a powerful method to model a large range of applications. The use of the multi-agent paradigm to model software systems has grown rapidly over the last few years. As a result, a variety of frameworks have been proposed to help in the development of multi-agent systems [3, 5, 6, 9, 21, 10, 2, 20, 7, 18]. All these frameworks aim to model agents to function in a range of applications as wide as possible. Consequently, they are more complex and difficult to implement for a specific application than frameworks that model agents to act in applications within a specific domain. The latter is the case of the framework developed in this work.

The reactive agents modeled in this framework react to stimuli that are generated when the value of one of their beliefs is changed. Depending on the agent rule in the system, each stimulus is mapped to different actions. Modeling an agent involves the definition of five entities: *sensors*, *beliefs*, *roles*, and *actions*. The properties of each entity and the relationships between them are described below:

- **Sensors:** they define how the agent retrieves sensorial information from its environment. Each sensor is associated with a belief representing an element of the environment.
- **Beliefs:** they model what the agents know about themselves and their environment. A stimulus is generated whenever the value of a belief changes.
- **Roles:** they specify how the agents must behave on the basis of their beliefs. They are responsible for mapping a given stimulus to an appropriate action.
- **Actions:** they implement the operations that must be performed for the agent to reach its objectives.

The life cycle of an agent takes the following sequence. The agent is initialized with initial values of the beliefs and perform a particular role in the system. The sensors continually observe an element belonging to the environment. When an element in the environment changes, the sensor updates the corresponding belief value. A stimulus is thereby generated and mapped to a particular action. Actions may be of a *personal nature*, altering one of the agent's beliefs or an element of the environment, or *collaborative*, resulting in messages to other agents. In this framework, agents communicated through the exchange of beliefs, which are forwarded within messages. When a message reaches

the recipient, the latter extracts the new belief and updates its set of beliefs. This new belief generates a new stimulus that results in a action from the recipient. Two types of belief are modeled: *data beliefs* and *action beliefs*. Data beliefs model what an agent knows about its current state and its environment. Action beliefs model the form of solving a particular problem.

2.2 Framework modeling

The description of the proposed framework follows the recommendations in [12].

2.2.1 Intent

The goal of this framework is to provide a set of reusable object-oriented classes that can be extended and customized for the implementation of information collection and publishing applications via the Internet, where processing involves solving mathematical programming models. Such models are very difficult to solve and polynomial time algorithms to solve them are not known to date, consequently all known algorithms may take a long time to finish.

2.2.2 Applicability

The domain of this framework comprises applications that need to solve several mathematical programming models to evaluate the situation of each team in the championship. The solution of each model involves a high computational cost, but each model can be solved independently in a different computer. Therefore, processing could be distributed and executed by computers with large computational power. Applications in this domain have a distributed nature, since the processing stage may be executed in several computers.

Since the sources of data are usually Internet web pages, the framework must take into account various features that may be present in some applications, such as: (1) the data source is not aware of the system, therefore the latter must be autonomous and search for the required data, (2) data may be inputed at any time and the updates should be done as quickly as possible; therefore the system must be capable of periodically checking the sources of data and automatically updating itself when data has been changed ed in any source, and (3) data collection can be very complex, involving tasks such as parsing an Internet page, which means that most of the applications in this domain use data validation to check for the existence of potential parsing errors.

2.2.3 Structure

Applications in the domain of this framework are dynamic, distributed, and autonomous. Bearing these features in mind, the framework was structured and modeled using the multi-agent systems software engineering paradigm. The framework can be divided into three subsystems responsible for collecting, processing, and publishing the data. Two types of agents implement the data collection subsystem: *Collectors* and *Validators*. The *Solver* agents are responsible for the processing subsystem. Finally, the *Publisher* agents implement the publishing system.

More than one agent of each type can compose each subsystem. For instance, the collection subsystem can be composed by two or more Collectors, each one collecting

game results from a different source to perform a consistent and safe parsing. Several Solvers, distributed over different computers, can compose the processing subsystem to handle data faster.

2.2.4 Participants

The four types of agents defined in this framework are:

- *Collectors* are responsible for retrieving game results from Internet.
- *Validators* are responsible for evaluating and validating the data sent by the Collectors.
- *Solvers* are responsible for generating and solving the mathematical programming models that provide qualification and elimination results.
- *Publishers* are responsible for combining the solutions of the mathematical models and publishing in the user interface.

2.2.5 Collaborations

The Collectors collect game results from Internet sources and send them to the Validator. If the data is consistent, the Validator sends it to the Solvers. Each Solver generates their mathematical models, solves them, and sends the results to the Publisher. The latter combines and publishes the results in a suitable medium in appropriate form for the application user.

2.2.6 Frozen spots

The frozen spots define elements of the agent infrastructure, such as:

- creation and initialization of the agents;
- definition of the agent beliefs;
- specification of the agent roles;
- implementation of reactive behavior of the agents; and
- infrastructure for agent communication.

2.2.7 Hot spots

The hot spots implement the behavior of each agent in response to details specific to each application, such as:

- implementation of the sensors according to the data source;
- implementation of the actions that will be performed by each agent; and
- implementation of the reasoning function of each agent, associating stimuli to actions according to the role of the agent in the application.

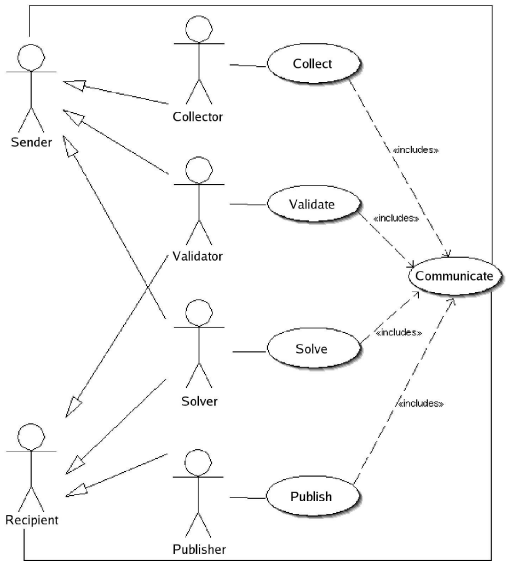

Figure 1: Use case diagram.

The framework proposed in this work is of the Gray Box type [11]: to generate an instance, it is not only necessary to implement the hot spots, but also to indicate the role and the initial values of the beliefs for each agent.

2.3 Framework design

Details of the framework design are presented using the UML language. Four actors take part in the framework use cases. Each of them represents one of the software agents defined in Section 2.2. The Collector, Validator and Solver actors are heirs of a Sender actor, since after their respective agents execute their tasks, they send the result to another agent. The Validator and Solver actors, as well as the Publisher actor, are also heirs of the Recipient actor, since the respective agents wait for data sent by other agents to carry out their tasks. Figure 1 illustrates the diagram of the framework use cases.

The *Collect* use case involves obtaining game results from one or more sources in the Internet (usually an HTML or XML page). The *Validate* use case involves assessing and validating the data sent by the Collectors and confirming that this data is consistent. In the *Solve* use case, data sent by the Validator is processed. The *Publish* use case involves combining the data sent by the Solvers and publishing the result in appropriate form to the application user. Finally, the *Communicate* use case involves sending a message containing a belief to another agent in the system.

To make the agents implementable in an object-oriented language, sets of classes need

to be created to represent the new entities related to the agents. This framework is composed by four modules that map the four entities making up an agent, and by a fifth module that implements the actions involving the exchange of messages between agents. Figure 2 presents the diagram of framework classes. The highlighted classes must be implemented in the framework instance. Sensors are implemented by the `AbstractSensor` and `ConcreteSensor` classes. Beliefs are modeled by the `AbstractBelief`, `DataBelief`, `ActionBelief`, and `ConcreateActionBelief` classes. Roles are represented by the `AbstractRule`, `Collector`, `Validator`, `Solver`, and `Publisher` classes. Actions are implemented by the `AbstractAction`, `CollectAction`, `ValidateAction`, `SolverAction`, and `PublishAction` classes. The message exchange module is composed of the `RmiServer`, `ReceiveMsg`, `AbstractProtocol`, `SendMsg`, `SendMultiCastMsg`, and `Message` classes, as well as of the `ReceiveMsgI` interface.

The design pattern *Observer* models the reactive behavior of the agents. When a belief is updated, the reasoning function of the agent is activated to process this event correctly. The classes involved are `AbstractBelief` (subject) and GenericAgent (observer). The stimulus mapping can be made by the *Command* design pattern.

3 Framework instance

The framework proposed in this work was used to build a system to automate the process of updating the web site of the FutMax project [24]. FutMax provides detailed information on the exact conditions of qualification and elimination of each team taking part in the Brazilian national soccer tournament and other soccer competitions. The reader is referred to Ribeiro and Urrutia [22, 23] for a detailed account of the application description and of the formulations of the associated integer programming optimization problems.

The Brazilian tournament is the most important soccer competition in the country and possibly the largest in the world, in terms of the number and quality of the participating teams. It is followed by millions of people who watch the games in stadiums, listen to radio broadcasts and watch the matches on TV. The characteristics of the tournament discussed below correspond to those of its 2004 edition. The tournament was organized in two stages. In the first, each team played against all others exactly once. In the second, the games are repeated in the same order, but with their home and away venues switched. A total of 552 matches were played between April and December.

A team receives three points for a win and one point for a draw. Teams are ordered in the classification table by the total number of points received. At the end of the competition, the first team in the table is the champion, the first four teams qualify for the Liberators of America Cup (an important South American tournament), and the four last teams are moved out of the division and will not compete in the tournament in the subsequent year.

Soccer is an important economic activity in Brazil. A team that drops down or fails to qualify for the Liberators of America Cup loses a lot of money. As a result, the team may be forced to sell its best players to cover its running costs, due to the loss of revenue from the sale of tickets and television rights. Consequently, the first goal of any team is to be champion, the second to qualify among the first four and, in the last case, to avoid the last four positions in the classification table.

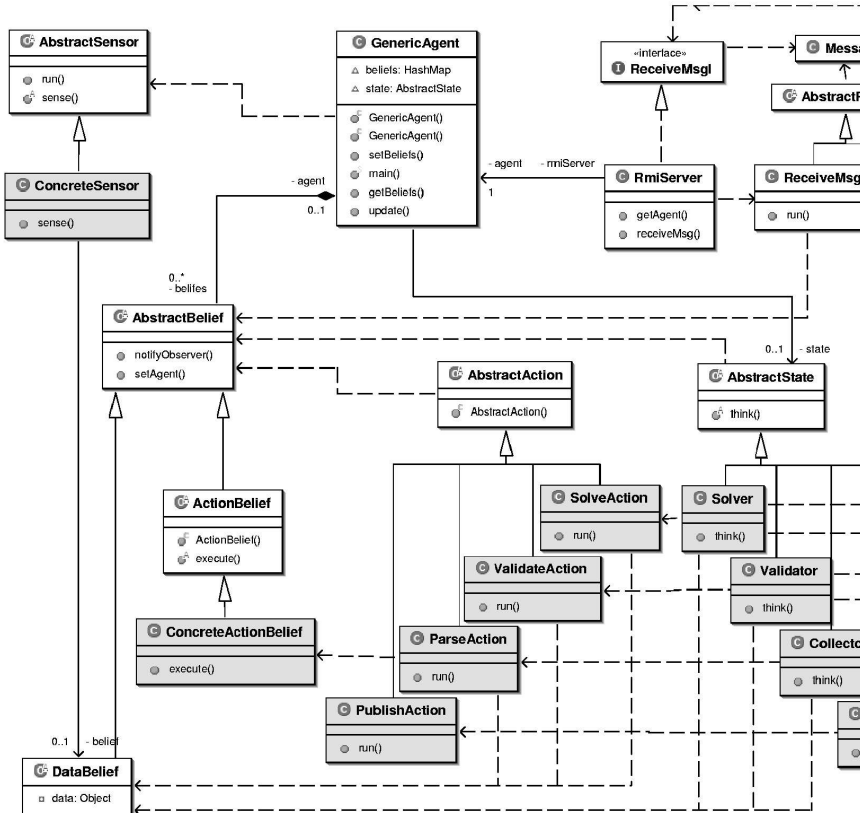

Figure 2: Class diagram.

3.1 Problem formulation

For a general tournament with the above characteristics, let n be the number of teams in the competition and m be the number of teams who qualify for the playoffs.

We denote by p_i the total number of points accumulated by each team $i = 1, \ldots, n$ at any particular moment of the competition, with $p_i = 0$ at the beginning. For any pair (i, j) of teams, with $i, j = 1, \ldots, n$ and $i \neq j$, let $g_{ij} = g_{ji}$ be the number of remaining games still to be played between teams i and j. At any time, a *valid assignment* is a set of triples $A(i, j) = (p_1(i, j), p_2(i, j), p_3(i, j))$ of non-negative integers for each pair (i, j) of different teams, such that $p_1(i, j) + p_2(i, j) + p_3(i, j) = g_{ij}$, $p_1(i, j) = p_3(j, i)$, and $p_2(i, j) = p_2(j, i)$, where $p_1(i, j)$, $p_2(i, j)$, and $p_3(i, j)$ represent, respectively, a possible number of victories of team i over team j, a possible number of games between team i and j which end up with a tie, and a possible number of victories of team j over team i, along the remaining g_{ij} games.

Given a valid assignment, the total number of points accumulated by each team $i = 1, \ldots, n$ at the end of the championship is

$$t_i = p_i + \sum_{j \neq i} 3 \cdot p_1(i, j) + \sum_{j \neq i} p_2(i, j).$$

For every valid assignment and in the context of the *Guaranteed Qualification Problem* (GQP), the final position of team $i = 1, \ldots, n$ in the standing table is defined as $P_i = |\{j : 1 \leq j \leq n,\ j \neq i,\ t_j \geq t_i\}| + 1$. Therefore, GQP for any team k consists in finding the minimum integer GQS^k such that for every valid assignment if $t_k \geq GQS^k$ then $P_k \leq m$ (i.e., if team k receives at least GQS^k points, then it will be qualified in the first m positions).

Similarly, in the context of the *Possible Qualification Problem* (PQP), the final position of team $i = 1, \ldots, n$ in the standing table is defined as $P_i = |\{j : 1 \leq j \leq n,\ j \neq i,\ t_j > t_i\}| + 1$. Therefore, PQP for any team k consists in finding the minimum integer PQS^k such that there exists at least one valid assignment leading to $t_k = PQS^k$ and $P_k \leq m$ (i.e., if team k receives at least PQS^k points, then it might be qualified in the first m positions).

The definitions of the position of a team in the standing table in the contexts of problems GQP and PQP are different. In the context of problem GQP, to ensure that team k is qualified despite any tie breaking rule, every other team with the same number of points as k is considered as qualified before the latter. Contrarily, in the context of problem PQP, we just consider the possibility (i.e., not the certainty) of qualification. In this case, team k has a chance to be qualified even if there is a tie with other teams in the first positions.

The scores GQP^k and PQP^k can be easily calculated for the cases in which a team k wins the competition, qualifies for the Liberators of America Cup and/or avoids relegation: being champion requires finishing in the top position in the table. Qualifying for the Liberators of America Cup requires finishing among the first four teams. Avoiding relegation, i.e., not finishing the competition among the last four places in the classification table, means finishing among the first 20 teams.

3.2 Integer programming model

The notation defined in the previous section is used to formulate an integer programming model for the *Guaranteed Qualification Problem*.

For any team $k = 1, \ldots, n$, let $\underline{GQS^k}$ be the maximum number of points such that there exists a valid assignment leading to $t_k \geq \underline{GQS^k}$ and $P_k > m$ at the end of the championship. Then, $\underline{GQS^k}$ is the maximum number of points a team can make and still not be qualified. Therefore, $GQS^k = \underline{GQS^k} + 1$ is the minimum number of points team k has to obtain to ensure its qualification in the first m positions. We define the following variables:

$$x_{ij} = \begin{cases} 2, & \text{if team } i \text{ has two wins over team } j, \\ 1, & \text{if team } i \text{ has one win over team } j, \\ 0, & \text{otherwise;} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{if } t_j \geq t_i \text{ (i.e. if team } i \text{ is not ahead } j), \\ 0, & \text{otherwise.} \end{cases}$$

The following integer linear programming model computes $\underline{GQS^k}$ for each team $k = 1, \ldots, n$, i.e., the maximum number of points team k can make and still not be qualified:

$$\text{GQP}(k): \begin{cases} \underline{GQS^k} = & \text{maximum } t_k & & \\ \text{subject to:} & & & \\ & x_{ij} + x_{ji} \leq g_{ij} & \forall\, 1 \leq i < j \leq n & (1) \\ & t_j = p_j + 3 \cdot \sum_{i \neq j} x_{ji} + \sum_{i \neq j}[1 - (x_{ij} + x_{ji})] & & \\ & & \forall\, 1 \leq j \leq n & (2) \\ & t_k - t_j \leq M(1 - y_j) & \forall\, 1 \leq j \leq n, j \neq k & (3) \\ & \sum_{j \neq k} y_j \geq m & & (4) \\ & x_{ij} \in \{0, 1, 2\} & \forall\, 1 \leq i \leq n, 1 \leq j \leq n, i \neq j & \\ & y_j \in \{0, 1\} & \forall\, 1 \leq j \leq n\,, j \neq k & \\ & t_j \geq 0 & \forall\, 1 \leq j \leq n. & \end{cases}$$

The objective function consists in determining the maximum number of points $\underline{GQS^k}$ team k can make and still not be qualified. Constraints (1) determine that only one team can win a game. Constraints (2) establish the total number of points obtained by each team at the end of the championship. For each team $j = 1, \ldots, n$, the sum $\sum_{i \neq j} x_{ji}$ gives the number of games won by team j among those still remaining to be played (three points each), while $\sum_{i \neq j}[1 - (x_{ij} + x_{ji})]$ corresponds to the number of games involving team j that end up with a tie (one point each).

Let M be an upper bound to the maximum difference between the number of points obtained by any pair of teams. Since there are 24 teams in the tournament and each of them plays exactly twice against every other, $M \geq 3 \cdot 46 = 138$ is a valid upper bound to $|t_j - t_k|$ for any pair (j, k) of teams, with $j, k = 1, \ldots, n$ and $j \neq k$. Constraints (3) state that if $t_j < t_k$, then $y_j = 0$ (i.e., team k is ahead j in the standing table). Constraint (4) enforces that team k is not qualified among the first m teams.

We now address the integer programming formulation of the *Possible Qualification Problem* for team $k = 1, \ldots, n$. Let PQS^k be the minimum number of points such that

there exists at least one set of valid assignments leading to $t_k = PQS^k$ and $P_k \leq m$ at the end of the championship. We define the following additional variables:

$$z_j = \begin{cases} 1, & \text{if } t_j > t_k \text{ (i.e. if team } j \text{ is ahead } k) \\ 0, & \text{otherwise.} \end{cases}$$

The previous model can be reformulated as follows to deal with the new situation:

$$\text{PQP}(k): \begin{cases} PQS^k = & \text{minimum } t_k \\ \text{subject to:} & \\ & x_{ij} + x_{ji} \leq g_{ij} \quad \forall\, 1 \leq i < j \leq n \quad (1) \\ & t_j = p_j + 3 \cdot \sum_{i \neq j} x_{ji} + \sum_{i \neq j} [1 - (x_{ij} + x_{ji})] \\ & \qquad \forall\, 1 \leq j \leq n \quad (2) \\ & t_j - t_k \leq M z_j \quad \forall\, 1 \leq j \leq n, j \neq k \quad (3') \\ & \sum_{j \neq k} z_j \leq m - 1 \quad (4') \\ & x_{ij} \in \{0, 1, 2\} \quad \forall\, 1 \leq i \leq n, 1 \leq j \leq n, i \neq j \\ & z_j \in \{0, 1\} \quad \forall\, 1 \leq j \leq n\,, j \neq k \\ & t_j \geq 0 \quad \forall\, 1 \leq j \leq n. \end{cases}$$

Constraints (3') play the same role as (3) in the formulation of GQP(k). They enforce that if $t_j > t_k$, then $z_j = 1$ (i.e., team j is ahead team k in the standing table). Constraint (4') states that there are at most $m - 1$ teams ahead k in the standing table. The infeasibility of PQP(k) means that team k is mathematically eliminated, i.e., it cannot qualify to the playoffs.

3.3 Hot spot implementation

The implementation details for each agent are presented below.

3.3.1 Collector agent

The data collection subsystem comprises three Collectors. The function of each Collector is to collect game results from an HTML web page, which presents the name of the teams and the result of each game played in the tournament to date. The Collectors read the same data from different web pages and send it to the same Validator agent. The latter compares the results and checks whereas at least two of them are equal. In this case, the Validator can assume that this result was read correctly.

Implementation of sensors: the only agent requiring a sensor is the Collector. Its sensor senses when an Internet page is updated. The sensor is responsible for keeping updated the belief value representing the date of the last alteration of the page that contains the game table. This sensor was implemented in the `ConcreateSensor` class, inherited from the `AbstractSensor` class, see Figure 2.

Collect action: an action belief is implemented for each Collector agent containing the specific algorithm for parsing its respective web page. This action results in a data belief that contains the game results.

Collector role: the reasoning function of a Collector maps two events. The first occurs when its sensor updates the belief value that stores the moment at which the web page containing game results was last updated. If the date of the last update is later than the date of the last parsing, the collector agent executes the collect action. The second event occurs when the collect data action generates a belief from game results. This event is mapped onto a send message action, where the collected data is sent to the validator agent.

3.3.2 Validator Agent

In addition to the collector agents, the data collection subsystem also includes one validator agent. The function of this agent is to validate the data collected by the collector agents. The collected data may present errors in the input of game results, as well as parsing errors resulting from changes in the web page structure. The application must anticipate these kinds of errors, given that the source of data is independent of the system.

Validate action: it compares the game results sent by Collectors. If two or more collectors have collected exactly the same data, then this data is considered to be correct (since each collector parses a different web page). This action results in a data belief that contains the validated game results.

Validator role: the reasoning function of a Validator maps two events. The first occurs when all Collectors finish sending their beliefs concerning game results. At this moment, the Validator executes a validate action. The second event occurs when the validate data action generates a belief from the validated game results. This event is mapped onto a send message action, where the validated data is sent to all Solvers.

3.3.3 Solver Agent

The processing subsystem comprises three Solvers. Each of them is responsible for solving, for each team, problems PQP and GQP regarding finishing in first place (becoming champion), finishing among the top four teams (Liberators of America Cup), and finishing among the first 20 teams (avoiding relegation). These agents are distinguished simply by the value of their beliefs, which specify which type of mathematical programming model should be generated.

Solve action: it builds and solves mathematical programming models associated to the PQP and GQP problems described in Section 3.2, for each participating team. This action results in a data belief that contains the scores PQP^k and GQP^k for every team $k = 1, \ldots, n$.

Solver role: The reasoning function of a Solver maps two events. The first occurs when the Validator sends the validated match results. This event is mapped onto a solve action. The second event occurs when the solve action generates a belief from the solution of the mathematical models. This event is mapped onto a send message action, where the belief is sent to the Publisher.

3.3.4 Publisher Agent

The publishing subsystem is formed by a single publisher agent responsible for three similar HTML pages, see Figure 3. The first page presents for each participating team the qualification and elimination figures related to winning the tournament. The second presents the figures related to qualification for the Liberators of America Cup. The third presents the figures related to relegation. Each page contains the following data:

- the name of each team,
- the number of games already played,
- the number of points already obtained,
- the teams mathematically qualified,
- the teams mathematically disqualified,
- the minimum number of points each team has to receive to be sure of qualification, regardless of any other results, and
- how many points each team has to receive to have a chance of qualification.

The situation of each team in terms of classification is displayed as a function of the font color of the team in the table. Teams displayed in green are those already qualified. Teams displayed in blue depend only on their own results to qualify. Those displayed in red depend on results of other teams to qualify. Teams in black have already dropped down (i.e., even if they win all their remaining games).

Publish action: it publishes a table in the form of an HTML page, containing the precise classification conditions for each team.

Publisher role: the reasoning function of a Publisher maps a single event that occurs when a solver agent sends the results of the mathematical models. This event is mapped to a Publish action.

4 Related work

There has been a lot of research on classification and elimination problems in soccer, hockey, and baseball tournaments [14, 4, 26, 25, 19, 1, 28, 8, 15]. Only two projects have a software system to maintain their pages on the Internet: Futmax and RIOT.

The RIOT project (Berkeley Remote Interactive Optimization Testbed) tracks the Major League Baseball (MLB), one of the biggest baseball competitions in the United States. The MLB is divided into two large leagues: *American* and *National*. Each league is divided into three subdivisions. In the first phase of the tournament, all teams within the same subdivision play each other. The second phase of the tournament involves playoffs between the top four teams from each league. The latter comprise the three first-placed teams from each subdivision and the best from the three teams that ended the first phase in second place. The winner of the playoff stage from the National League plays the winner

of the American League in the grand finale of the tournament. As in many other sports competitions, MLB fans are always eager for information on the situation of their favorite teams in terms of their qualification and elimination chances for the next stages of the tournament.

The system maintaining the RIOT project site is fully automated and functions as follows: every day, a free Internet news release service (www.infobeat.com) sends an e-mail containing the results of the games played the night before. The process of updating the site is initiated at precisely 2 a.m. when the game results contained in the e-mail sent by Infobeat are parsed. The game results from the day in question are added to a text file that contains the results of all games already played. Next, the latter is used to generate optimization models, which are solved by CPLEX [16]. Finally, the CPLEX output file is parsed to extract the solutions from the mathematical modules. These solutions are then used to update the web page. Occasionally, the daily e-mail containing the game results may be delayed: in this case, the program must be re-initiated manually by an operator as soon as the e-mail arrives.

The FutMax system proposed in this work handles the main drawbacks encountered in the RIOT system. Data collection is executed automatically. The collector agents are autonomous and periodically check the pages announcing the game results. As soon as these pages are updated, the Collectors parse the game results and send them to the validator agent. This eliminates the need for an operator and the dependence on a service that sends the game results by e-mail. Futmax is less prone to failures, since it includes various agents collecting the same data from different sources and an agent to validate the parsed data. Data processing in FutMax is more efficient, since it is divided into three solver agents that process the information on different computers, thereby increasing the processing power and the quantity of memory available. Due to its implementation in Java, FutMax2004 is the most portable alternative. In addition, FutMax is fully autonomous and does not require human inputs in the course of the tournament.

5 Conclusions and extensions

We presented in this work a framework for implementing applications related to qualification and elimination problems in sports tournaments. Due to the fact that the framework is implemented in Java, using the Java Concert Technology API [16], the framework enables the quick development of portable applications. Agent-oriented modeling enables the development of a highly uncoupled systems, because each agent represents each subsystem and they are connected only by the exchange of messages. Moreover, communication between agents through the exchange of messages renders the distributed nature of the application transparent.

An instance of the framework, created to follow the Brazilian soccer tournament, was shown to be far superior to similar applications found in the literature. The framework enables the development of autonomous applications that need few human interactions. The concept of cooperation between agents enables a relative simple and safe solution to the problem of collecting data from sources on the Internet that are not connected to the application. Since the solution of the integer programming models can be done in a distributed manner in several computers, the response of the system is faster than the other approaches presented in the literature.

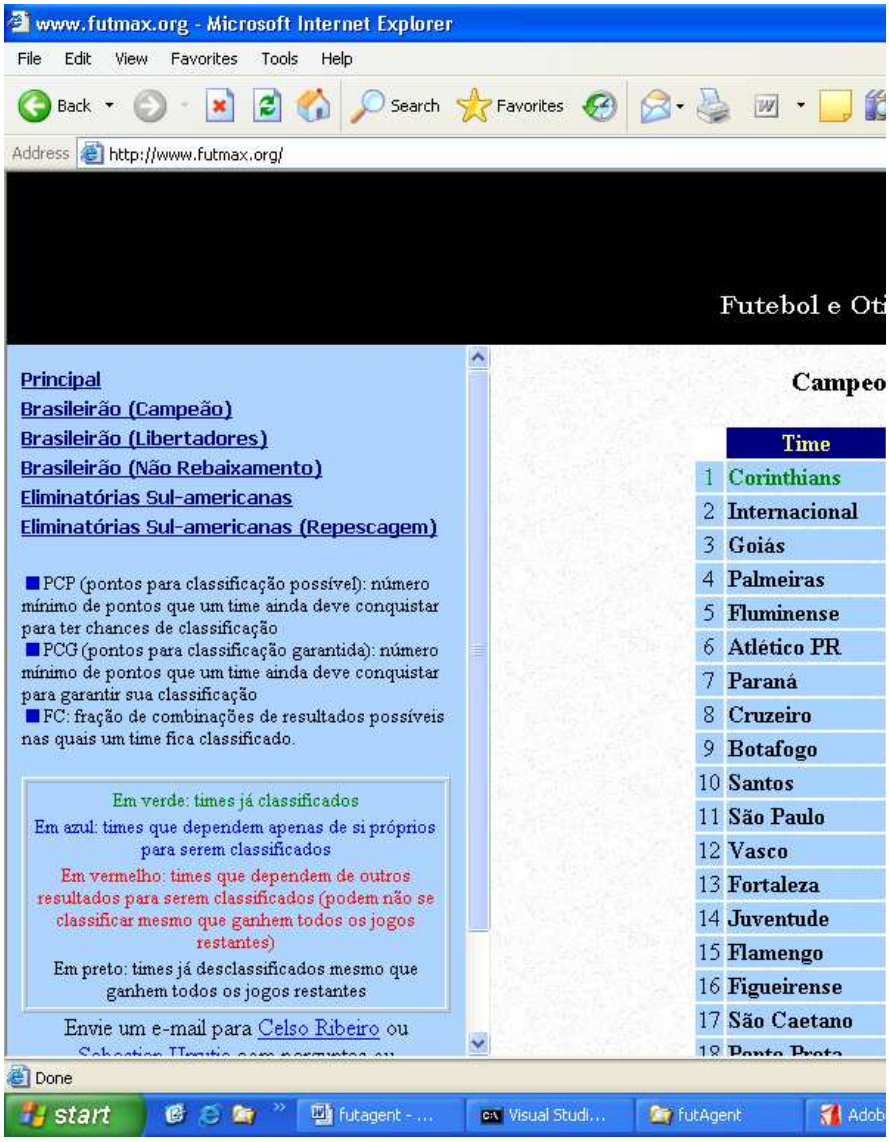

Figure 3: Use case diagram.

References

[1] I. Adler, A.L. Erera, D.S. Hochbaum, and E.V. Olinick. Baseball, optimization, and the world wide web. *Interfaces*, 32:12–22, 2000.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - A FIPA-compliant agent framework. In *Practical Application of Intelligent Agents and Multi-Agents*, pages 97–108, London, 1999.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience*, 31:103–128, 2001.

[4] T. Bernholt, A. Güllich, T. Hofmeister, and N. Schmitt. Football elimination is hard to decide under the 3-point rule. In M. Kutylowski, L. Pacholski, and T. Wierzbicki, editors, *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, volume 1672 of *Lecture Notes in Computer Science*, pages 410–418. Springer, 1999.

[5] J.M. Bradshaw, S. Dutfield, P. Benoit, and J.D. Woolley. KAoS: Toward an industrial-strength generic agent architecture. In J.M. Bradshaw, editor, *Software Agents*, pages 375–418. AAAI Press, 1997.

[6] F. Brazier, B. Dunin-Keplicz, N.R. Jennings, and J. Treur. Desire: Modeling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.

[7] D. Chauhan and A.D. Baker. JAFMAS: A multiagent application development system. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 100–107, New York, 1998. ACM Press.

[8] E. Cheng and D.E. Steffy. The Hockeyplex project. Online reference at http://personalwebs.oakland.edu/~desteffy/hockeyplex/Hockeyplex.pdf, last visited on May 28, 2005.

[9] J. Collis and D. Ndumo. Zeus technical manual, 1999. Intelligent Systems Research Group, British Telecommunications.

[10] R.S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughanam. Jackal: a Java-based tool for agent development. In *AAAI Workshop on Tools for Agent Development*, pages 73–83, Madison, 1998.

[11] M.E. Fayad, D.C. Schmidt, and R.E. Johnson. *Building application frameworks - Object oriented foundations of framework design.* Wiley, New York, 1999.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley, 1994.

[13] A. Garcia and C. Lucena. Software engineering for large-scale multi-agent systems. *ACM Software Engineering Notes*, 27:82–88, 2002.

[14] D. Gusfield and C.E. Martel. The structure and complexity of sports elimination numbers. *Algorithmica*, 32:73–86, 2002.

[15] A.J. Hoffman and T.J. Rivlin. When is a team 'mathematically' eliminated? In H.W. Kuhn, editor, *Proceedings of the Princeton Symposium on Mathematical Programming*, pages 391–401. Princeton University Press, 1970.

[16] ILOG. ILOG CPLEX 8.0 user manual, 2002.

[17] N. R. Jennings. Agent-oriented software engineering. In F. J. Garijo and M. Boman, editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering*, volume 1647 of *Lecture Notes in Computer Science*, pages 1–7. Springer-Verlag, 1999.

[18] K. Kendall, P. Krishna, C. Pathak, and C. Suresh. A Java application framework for agent based systems. In T. Lacey, editor, *ACM Computing Surveys Symposium on Application Frameworks*, Air Force Institute of Technology, Dayton, 2000.

[19] S.T. McCormick. Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Operations Research*, 47:744–756, 2000.

[20] H.S Nwana, D.T. Ndumu, L.C. Lee, and J.C. Collis. ZEUS: A toolkit and approach for building distributed multi-agent systems. In O. Etzioni, J.P. Müller, and J.M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents*, pages 360–361, Seattle, 1999. ACM Press.

[21] S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: open source for open standards. In *5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 355–368, Manchester, 2000.

[22] C.C. Ribeiro and S. Urrutia. OR on the ball: Applications in sports scheduling and management. *OR/MS Today*, 31:50–54, 2004.

[23] C.C. Ribeiro and S. Urrutia. An application of integer programming to playoff elimination in football championships. *International Transactions in Operational Research*, 12:375–386, 2005.

[24] C.C. Ribeiro and Sebastián Urrutia. Projeto futmax. Online reference at http://www.futmax.org, last visited on March 23, 2005.

[25] L.W. Robinson. Baseball playoff eliminations: An application of linear programming. *Operations Research Letters*, 10:67–74, 1991.

[26] B. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8:302–308, 1966.

[27] V. Silva, A. Garcia, A. Brandão, C. Chavez, C. Lucena, and P. Alencar. Taming agents and objects in software engineering. In A. Garcia, C. Lucena, J. Castro, A. Omicini, and F. Zamboneli, editors, *Software Engineering for Large-Scale MultiAgent System*, volume 2603 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 2003.

[28] K.D. Wayne. A new property and a faster algorithm for baseball elimination. *SIAM Journal on Discrete Mathematics*, 14:223–229, 2001.

[29] M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: the state of the art. In P. Ciancarini and M. Wooldridge, editors, *First International Workshop on Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 1–28. Springer-Verlag, 2001.