Accomplishing Adaptability in Simulation Frameworks: the Bubble Approach

J. Andrés Díaz Pace[1,2], Federico U. Trilnik[1] and Marcelo R. Campo[1]

[1] *ISISTAN Research Institute, Facultad de Ciencias Exactas,*
Universidad Nacional del Centro de la Provincia de Buenos Aires
Campus Universitario, Paraje Arroyo Seco, B7001BBO Tandil (Argentina)
Email: { adiaz , ftrilnik, mcampo } @exa.unicen.edu.ar

[2] *Also CONICET*

Abstract. Enforcing framework adaptability is one of the key points in the process of building an object-oriented application framework. When it comes to simulation, some adaptation mechanisms to configure components on-the-fly are usually required in order to produce good software artifacts and alleviate development effort. The paper reports an experience using a simulation multi-agent framework, initially conceived to be used in fluid flow problems. The framework architecture demonstrated during its evolution a great potential regarding to flexibility and modularity, tackling a wide range of other problems ranging from a network protocol simulation to a soccer simulation.

Keywords: multi-agent systems, object-oriented application frameworks, simulation, adaptability.

1. INTRODUCTION

Object-oriented application frameworks are usually regarded as a useful technology to achieve reuse in software systems [Fayad97]. The benefits of a framework [Johnson97] are that it provides a general and reusable skeleton of classes and behavior patterns for a given domain, and relying in this support new applications can be developed in a flexible and direct way, with additional savings of time and design effort. But this is only one side of the coin, these benefits should be enforced during design time to effectively be successful accomplishing well-sound software artifacts. Besides, the process of building a framework is far to be straightforward. It often requires important efforts to capture a given domain abstraction and provide a powerful and comprehensible framework to application developers. A common methodology [Fayad99] consists of an iterative development, starting with a few examples and applying successive refactorizations to the framework until it reaches a reasonable state of maturity. Interestingly, this process sometimes leads to a framework suitable for other problems not foreseen by developers during the first steps of design. One of the causes of these variations can be a misunderstood framework usage due to inexpert users trying to fit their specific needs. If the framework can take profit of these “abnormal” situations, new unexpected capabilities can be included producing an evolutionary leap in the framework.

The paper presents an object-oriented framework implemented in Java (named Bubble) designed under the multi-agent approach [Demazeau91, Drogoul92], originally conceived to simulate the motion of gas bubbles in a fluid environment. Later developments and applications demonstrated the wide potential of the framework, which progressively became a flexible support to define and organize simulations of cooperative processes characterized by the interaction of large numbers of individuals. Moreover, it was applied to model other problems not expected at the beginning, such as a network protocol simulation and a soccer simulation. As a result, novel possibilities to extend the original framework were discovered and analyzed. A report of these experiences is also included in the article.

The contribution of the work is that it provides an adaptable framework architecture with a particular combination of building mechanisms such as agents, uniform decomposition, competing tasks, events and implicit invocation, to represent complex simulations in a flexible manner. In addition, it describes three application examples using the framework, showing both practical experience about framework evolution and derived research ideas in the targeted area.

The work is organized into five sections. The first section gives background information about adaptability in object oriented systems and multi-agent systems. Then, the description of our framework Bubble is presented. The following section reports three application examples based on the framework and lessons learned in this evolution process. Then, some tradeoffs and perspectives about Bubble are discussed after that. And finally, we draw the conclusions of the work.

2. BACKGROUND INFORMATION

This section explains what we mean by adaptability in object oriented systems (closely related with the architecture of our framework), and presents basic concepts about multiagent systems and agent-based simulations, in order to situate the rest of the work.

2.1. Adaptability in object-oriented systems

Adaptability is an important and desirable quality factor in today's software system. This property defines the ability of a given software system to cope smoothly with changes in the problem specification, producing a low impact on components previously implemented [Fayad96]. In this way, systems are able to evolve and tackle different variations of a given problem.

From a high-level point of view, a software artifact usually needs two properties to be adaptable: extensibility and flexibility. By extensibility, we mean the ability to change capabilities of the system in amount (e.g. using inheritance), whereas flexibility is to change capabilities in kind. The lack of adaptability can become a critical issue if you want to scale up the system. Developers should not only think in the target system during the design phases, but in the future variations of such system as well (for example, emphasizing software decentralization and modularity).

These two aspects, extensibility and flexibility, are especially useful in modeling problems, because we usually start with a quite abstract description of our problem, and then new features are included in this description and different variations are tested. Therefore, the development effort associated with the modeling process can be an important factor, even having some kind of component reusability. It requires some adaptation mechanisms to configure components on-the-fly, in order to produce good software and alleviate development effort. As a drawback, this gain in expressiveness and adaptability of the system often is balanced with some loss in performance.

The building mechanisms included in Bubble's architecture that we will describe in the next section, try to enforce adaptability issues to easily configure simulation applications.

2.2. Multi-agent systems

With the spreading of the agent paradigm [Demazeau91, Sycara98] as a derivation of the object orientation paradigm [Meyer97], a new way of thinking and building complex software is increasingly taking place. Under this new paradigm, software systems can be conceived as organizations of interrelated agents, with flexibility and modularity as major potential gains.

An agent is a computational entity evolving in an environment, with an autonomous behavior, capable of perceiving and acting in this environment, and capable of communicating with other agents. A multi-agent system is a set of agents, probably with some organization, interacting in a shared environment. The research in multi-agent systems is centered in analyzing how a collection of autonomous agents can solve a given problem that usually is beyond the scope of individual capabilities.

The main advantages of a multiagent system over a single and monolithic system [Moulin96] are the following:

- Higher capability for problem solving because of the possible parallelism,
- Flexibility, because agents with different capabilities associate to solve a given problem.
- Robustness, because control and responsabilities distributed among the agents result in a better fault tolerance.
- Scalability, because proper agent modularity makes easy to add new agents with new capabilities in the system and program different agents.

As regards multi-agent simulation models [Drogoul92], a mapping from each component of the real system (individual or group) to an equivalent computational agent is defined, and the simulation is based on the global consequences of local interactions between members of the population. These models typically consist of an environment in which the interactions occur, and some number of individuals defined in terms of their behaviors (procedural rules) and characteristic parameters. Individuals might represent plants and animals in ecosystems, vehicles in traffic, people in crowds, etc.

3. THE FRAMEWORK BUBBLE

Bubble is a multi-agent framework implemented in Java [DiazPace99], originally conceived and designed to simulate the motion of gas bubbles in a fluid environment. The basic elements of the system are reactive agents [Demazeau91] described by an internal state and a set of executable tasks. The interaction among these reactive agents is performed through events that the agents produce and receive. The agents are equipped with associated sensors (like filters) that are registered to hear certain kinds of events with a defined criterion of relevance (local, by group, by event strength, regional, etc.).

The behavior of a reactive agent is defined through tasks using a condition-action style, i.e. a task is a module composed by a series of actions to be executed by the agent (action part) when certain conditions are fulfilled (condition part). Conditions can be related either to the internal state of the agent or the incoming events. Note that these tasks should not be confused with statecharts. The framework also admits agents containing groups of other agents, and tasks composed by groups of predefined tasks. In this way, complex interactions, structures and behaviors can be modeled combining primary blocks.

Figure 1 shows a diagram of the conceptual model supported by the framework, illustrating a typical event flow between agents in a container and the role that sensors play in this process. Note that the outgoing events produced by agent D are propagated only if the agent is attached to a container, but this relationship is not compulsory. When an agent receives an incoming event (agents B, D and container, in the example), the processing depends on the current tasks associated with the agent.

The framework can be described from three different perspectives: structural organization, communications, and agent tasks. Each of these views refers to a group of collaborating classes in the framework, and the following sections provide more details about them.

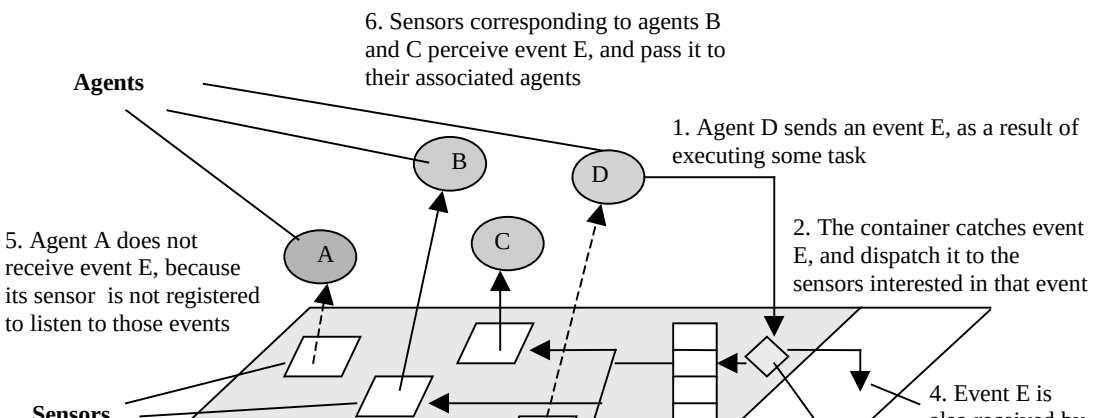


Fig. 1. Conceptual model of Bubble´s architecture

3.1. Structural organization

Bubble is organized applying the paradigm of uniform decomposition. By uniform decomposition [Bass98] we mean the operation of separating a large component into two or more smaller ones, limiting the composition mechanisms to a restricted uniform set. Thus, integration of components and scaling of the system as a whole is achieved, having besides modifiability and reusability properties. The aim is to represent the agent organization with a hierarchy of abstraction levels: agents composed by other agents, that in turn are composed by others, and so on. The same structure is used to handle incoming and outgoing events. As example, we can apply these concepts in a simple prey-predator model: a container-agent might represent the environment where both preys and predators live, and a composed agent can be used to model a gang of predators looking for new preys.

3.2. Communications

Communications among different components in Bubble are performed through events. Every agent can be linked to a container-agent, and this container is engaged to collect and dispatch incoming events to the sensors registered inside it. As we explained in a previous section, sensors acts like filters and transmit only interesting events to their associated agents. An implicit-invocation mechanism [Shaw96] is used to achieve these notifications. The container-agent is in charge of the event flow management among all the agents.

An event represents a notification of any change occurring in the system. Sensors are responsible of reception and conditional transmission (filtering) of events. Container-agents deliver events received from the agents to the sensors. To illustrate this interaction, suppose we are modeling a market where buyers and sellers are free to perform transactions, any customer interested in buying certain items needs to specify a purchasing criterion and enroll itself with the market to listen to bids. In this context, the market can be a container-agent, both sellers and buyers are simple agents, and the specific purchasing criterion corresponds with a sensor. Every time a new offer appears, our customer will be notified about that situation only if the offer fulfills its purchasing conditions.

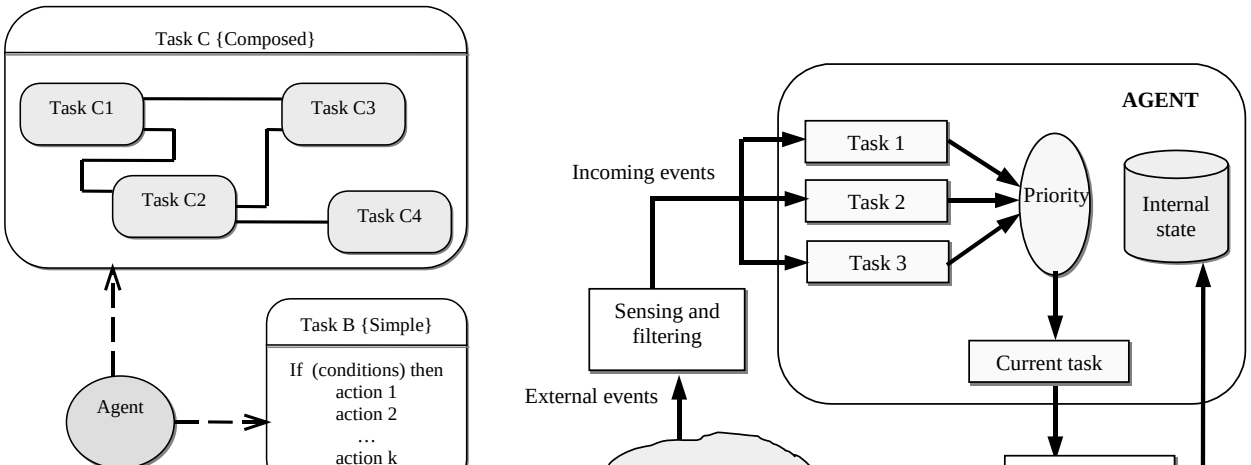


Fig. 2. Competing tasks in the framework

3.3. Tasks

All the agents of the framework can perform a set of tasks. A task is composed by one or more procedures with a set of input and output parameters. Tasks are triggered by predefined conditions, which can be related to the internal state of the agents or incoming events. In this way, the agent behavior is conceived as a set of competing tasks, where only one task is active at the same time [Drogoul92]. When a selected task is executed, it can generate either outgoing events and/or changes affecting the agent state. Figure 2 provides a picture of such situation. Different tasks can be dynamically assigned to an agent, and they compete to execute according to their priorities and activation requirements.

4. CASE STUDIES

This section reports three application examples using Bubble, ranging from the early examples modeling fluid flow problems, the example representing a network protocol simulation, and the more recent development modeling a soccer simulation. These case-studies are presented in the chronological order they were developed, showing how the original framework was adapted to fulfill the requirements of the different problems. More interestingly, it reveals how the building mechanisms included in Bubble allowed us to achieve a remarkable adaptability.

4.1. Bubbly flow simulation

Bubbly flow [Herrero96] is encountered in many industrial applications, such as distillation columns, nuclear and chemical reactors, oil piping, among others. Wherever two unmiscible fluids are forced to flow together, one of them tend to concentrate in bubbles, the other fluid acting as a continuous carrying environment. Generally, some macroscopic global magnitudes are used to characterize the state of the flow, representing the relative amount of each fluid component, the number of bubbles per unit volume, the interfacial area density, the average bubble size, etc.

Classically, complicated sets of partial differential equations are used to describe the rate of change and spatial distribution of the global magnitudes. Afterwards, the computer is used to numerically solve the field equations. Instead, the multi-agent modeling introduces the computer at the beginning of the

description, simulating the movements and changes of the fluid particles, and afterwards global statistical patterns are identified to determine the general laws.

4.1.1. Implementation using the framework

We represent the bubbly flow as a multi-agent virtual world composed of a continuous liquid which will be partitioned in slices to construct spatial geometries, and a disperse phase instantiated in numerous bubbles embedded in the liquid. The bubbles are codified as reactive agents represented by spheres, which perform the following set of tasks: (a) Displacement, the center of each sphere is displaced a constant distance d in a random direction θ; (b) Coalescence, when two or more bubbles collide, they coalesce giving birth to a new bubble conserving the total volume; (c) Breakup, each bubble is allowed to break up into two bubbles conserving the total volume, with a given probability model (for example when the size is larger than a certain critical size).

The continuous fluid is partitioned in slices. These slices, being agents themselves, have particular properties, such as turbulence intensity represented by spatial variations in the displacement task. By linking the slices, different geometrical configurations can be constructed, such as pipes, bends, or irregular vessels. The slices provide the figure of a neighborhood, precluding interactions between distant bubbles, by limiting the reception of inner events to bubbles located within the corresponding slice. This feature greatly reduces the flow of events in the system. Additionally, complicated geometries can be easily simulated by partitioning the space in blocks represented by slice agents. Figures 3 shows a diagram of this implementation. The side effect is the tracking and accounting of bubbles moving from one slice to another.

The bubbly flow model was implemented specializing the Bubble framework according to the requirements of the application domain. The model was defined by an environment agent, divided in slice agents representing the continuous fluid. Inside every slice, bubble agents represent the dispersed fluid. The agents Environment and Slice are modeled as specializations of container-agent. The visualization of the simulation is managed by an additional visualization-agent, and the statistic data is collected and processed by the an storage-agent. These last two components were implemented following the same conceptual model defined in the framework: the visualization agent listens to events about creation and deletion of bubbles updating a canvas in consequence, whereas the statistical agent catches events to compute statistical indicators that stores in an output file.

Moreover, redefining the bubble agents tasks, more complex transport phenomena can be appropriately simulated with great easiness. Likely, composed tasks can be engineered to resemble complex interactions between bubbles, such as wake trapping and vortex induction. On the other hand, mass exchanges between phases are simulated by tasks performed by Slice and Environment classes, which can create and annihilate bubble instances. At that stage, a concurrent implementation of the framework was still pending.

4.1.2. Results and lessons learned

Fluid flow problems were the basis of the framework. The framework tried to capture commonalties of this domain and emphasize easy-of-modeling features. The following items summarize the main results of this stage.

- *Uniform decomposition*. The notion of contained and container agents provides different levels of abstraction to represent entities involved in the simulation (i.e. bubbles, container fluids and bubble sources). It also permits future refinements in this modeling hierarchy.
- *Event notification and filters*. Event notification resulted a good mechanism to integrate different entities into the model in a step-by-step fashion. Afterwards, sensors and containers were introduced as filters to manage event flow.

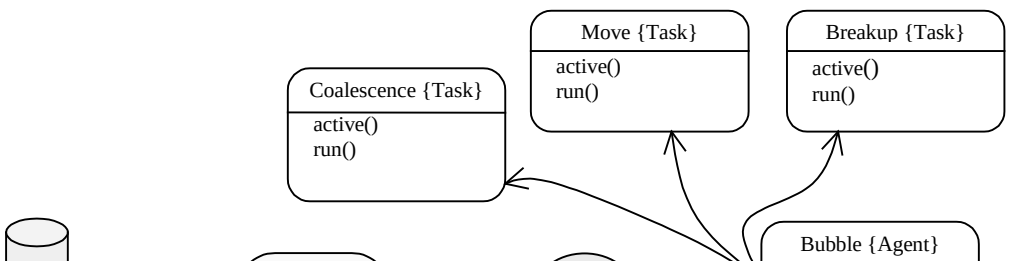


Fig. 3. Implementation of the bubble flow simulation

- *Simple tasks.* Most of the tasks we defined for the agents were quite simple at this stage, and they usually encapsulated equations ruling geometrical laws (i.e. movements, breaks, coalescences, and rebounds).
- *Geometry*. The physical nature of the simulation lead to define most of the filtering criteria based on geometrical notions. Initially, the problem was implemented and tested in a 2D world, with geometrical considerations quite tangled into multi-agent code. After that, we built a 3D model for the simulation.
- *Utilities.* Using the same architectural ideas we implemented visualization and statistics facilities.

As we mentioned, the original 2D fluid model was extended to 3D, and other new components were included into the model. For instance, we inserted bubble sources and sinks to represent variations in the volume of bubbles. The impact of these changes was minimum, only affecting calculus related with distance considerations. Moreover, the 3D extension was accomplished with a relatively small additional work. This capability to make smooth transitions from one initial model to more elaborated ones, was one of the most relevant aspects of the framework architecture. However, we observed in the

implementations that good flexibility and extensibility often came with some tradeoffs regarding to event control and efficiency.

4.2. Internet mobile host protocol

The protocol is designed to route packets between mobile hosts [Perkins94], i.e. portable computers connected to different networks probably changing their locations as they are operating. A unique and permanent host location (called home location) is assigned to every host. There is also a routing task, which delivers packets sent to the host home address towards the current location of the host in a transparent and efficient way. A simulation of this protocol was implemented using our framework to study different parameters in the process.

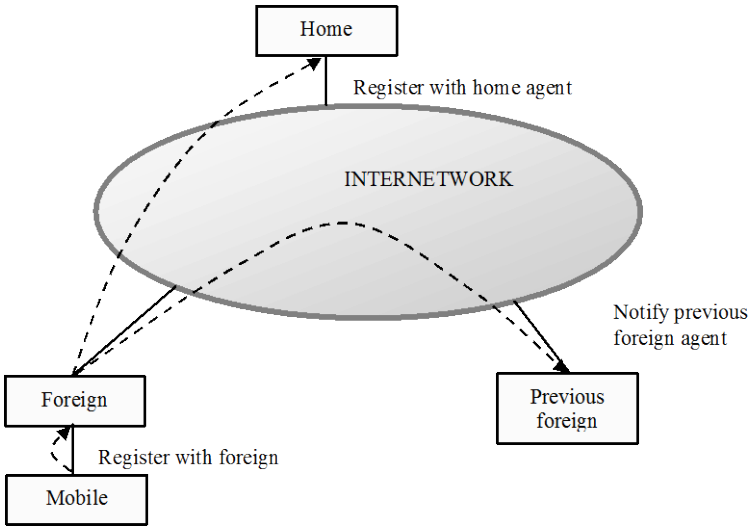


Fig. 4. Registration process in the Internet Mobile Host Protocol

4.2.1. Implementation using the framework

We suppose our world divided in several networking areas (e.g. LANs, wireless cells, or other kind of networks). Each area contains: a foreign-agent with a registry of every mobile host visiting such area, and a home-agent storing the hosts whose home location belongs to the area but they are currently out of its scope. If a new host arrives to the area, it must be registered by the local foreign-agent, as it is shown in Figure 4. Any packet sent to a mobile host is routed to the host home LAN, and this packet is then caught by the home-agent. The home-agent checks the current (temporary) address of the destination mobile-host, forwarding the packet to the respective foreign-agent when it is necessary. Thus, the sender host is notified of the situation and the next packets are directly sent to the current host address. Some mobile-hosts can act as cache-agents, storing temporary information about real addresses of other mobile-hosts.

The simulation was implemented modeling mobile-hosts, home-agents and foreing-agents as simple agents, and defining some container-agents to represent the home-networks and the Internet (i.e. a network including all the other home-networks). Each mobile-host has an IP address and different tasks associated. These tasks involve capabilities such as: sending packets, moving around Internet and registration with foreign-agents. A home-agent was specialized to manage a home-list, pass packets to hosts, and route packets to foreign-agents when it is necessary (notifying also to the specific sender host). In a similar way, a foreign-agent can announce its operation to allow other hosts to be registered, maintain a list of visitor hosts, and resend packets from other sources. When it comes to networks, their

activities include: capturing packets in their domains and routing these packets to other interconnected networks. As a result, mobile-host agents wander through different network areas.

4.2.2. Results and lessons learned

With this second example we started to think a little beyond, not only centered on fluid problems. This new point of view lead us to some changes in the framework, which we detail below:

- *Geometry*. In this network simulation, we realized that geometrical aspects may or may not be required by some agents, depending of the relative importance of a topological structure in the problem. For instance, networks only need a containment criterion based on their area of influence, and hosts do not care about geometrical considerations. In consequence, we could consider geometry as an enabled/disabled feature.
- *Groups.* Composed agents became more important to represent groups of agents, without imposing a strict containment relationship among agents.
- *Multicast and broadcast events.* Multicast and broadcast messages, closer to network domains, were also required as a way to provide more elaborate mechanisms for agent communication.
- *Utilities.* Tasks such as visualization, statistic collection and inspection became quite complex. To face this situation, we believe that a general toolkit supporting these facilities is a potential requirement to be included in the framework.

4.3. Soccer

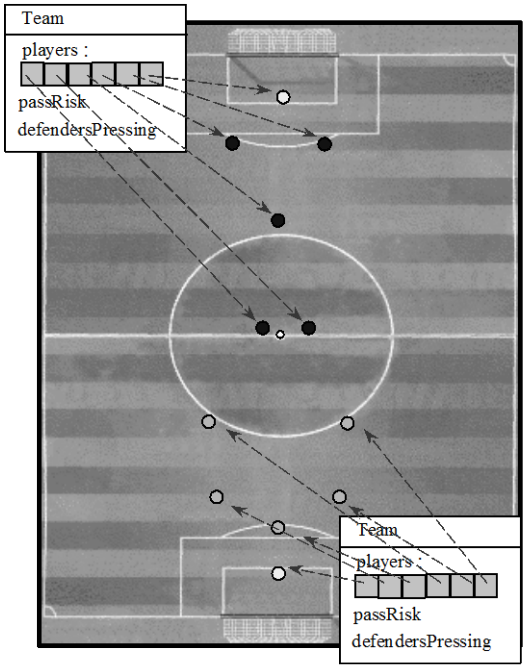
Nowadays, soccer games are used as a testbed to analyze and evaluate performance of several systems based on artificial intelligence approaches. Soccer usually includes classic characteristics encountered in artificial intelligence research such as: players having incomplete information about world states, a continuous and unpredictable game that cannot be planned in advance, or players facing situations that should be solved using their individual capabilities in conjunction with team collaboration [Stone98, Stone99, També97]. Any soccer game (even simplified) offers a wide variety of situations requiring some kind of coordination mechanisms. The term coordination refers to the process where each agent "reasons" (in a more or less complex manner) about its actions and the (anticipated) actions of other agents in the community, trying to achieve a coherent behavior in the group. For instance, avoiding all the players running for a given ball, accomplishing predefined movements, coherent defending techniques, corner execution, are examples of coordination situations.

In this context, it is of particular interest to investigate coordination techniques simple enough to be implemented by reactive agents. Focused on the study of such mechanisms, we decided to develop a soccer simulation using the support provided by Bubble's architecture [Campo99].

4.3.1. Implementation using the framework

The simulation models a game between two teams, where each player is represented as an agent with certain capabilities to solve different situations arising during the soccer game (see Figure 5). In order to concentrate the main efforts on coordination issues, we made some assumptions such as: a 2D-game, no human-like movements in players, each player having an unlimited view of the field, no blocking actions among players, and no off-side situations. The more relevant player capabilities are the following: stopping a ball, running with a ball, passing a ball to another player, shooting, taking a rival, looking for an offensive position, moving to a defending zone, and so on. A player behavior can be different according to its role (e.g. goalkeeper, forward, defender, or middle-player). Individual characteristics of the players can be changed through time. Changing characteristics provide a well-sound support to define and implement different team strategies.

We considered any actor in the system as a reactive agent. Therefore, a player, the ball and the referee are considered as agents in our simulation. All these agents live inside a field, that we represented with a container-agent. Agent behaviors are defined as different tasks; a player agent can, for instance, go straight to a ball, mark, or take a kick. These tasks trigger events to notify other agents about changes in the game state, and these events are also perceived by the rest of the agents via their sensors. These mechanisms are implemented following the prescriptions of Bubble architecture. Furthermore, there are other special tasks related with collaboration to solve situations presented during the game (e.g. selecting the best passing in a given play). These situations involve special events that are explicitly sent by agents with a communication purpose. We implemented a common repository (called map) to publish global data about the world (for instance, player positions or ball location), in order to facilitate the communication processes. It also provided an easy and more efficient way of sending messages from players to the ball, because the original mechanism based on sensors and events would be a potential point of data overflow. As side effect, the debugging and traceability of the application might be simplified.



In this example, a team structure groups a set of players and defines some properties like: *passRisk* property (estimation of a bad passing) and *defendersPressing* property (the number of players pressing on a ball in a given situation).

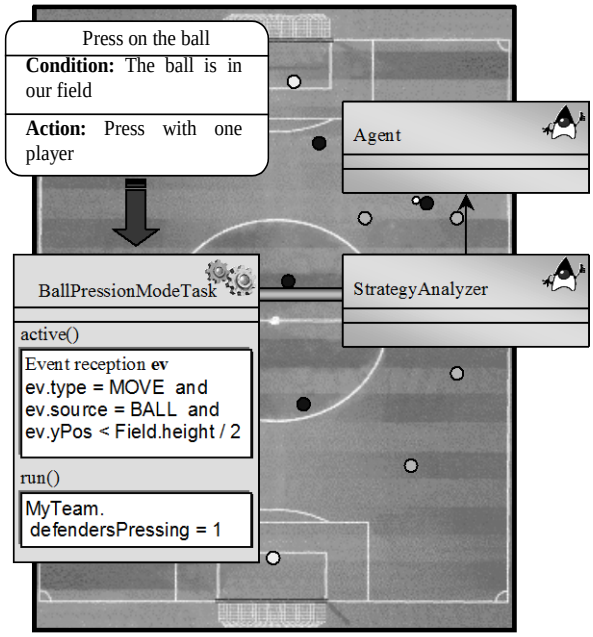
Fig. 5. The *Team* structure determines basic player positions during the game

A coach agent was introduced to manage group formations. Using this metaphor, players belong to a given team, and there is master agent who is able to configure player properties before a game (i.e. offensive positions, passive positions, or player strengths). Actually, our simulated coach was not a real cognitive agent: its directives came from predefined actions provided by users during the game using a GUI panel (see Figure 7). Despite of this simplification, this conceptual model can include more elaborated coach reasoning in future developments (some alternatives are a case-based reasoner or some kind of planning).

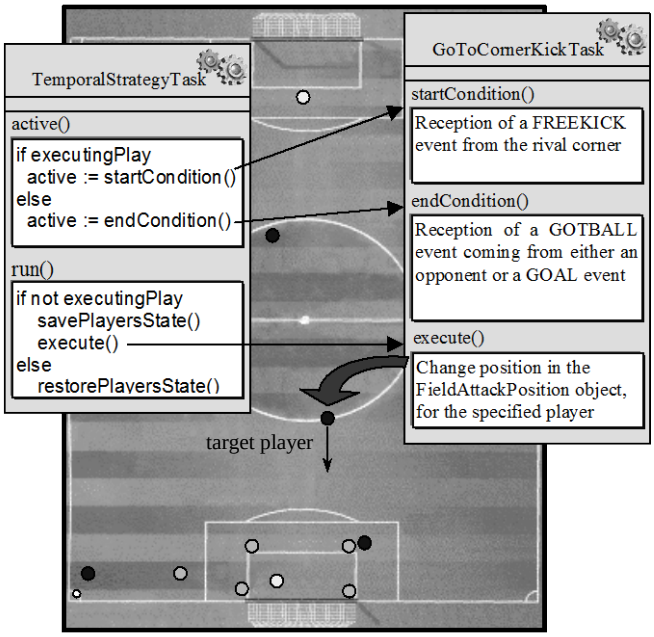
As regards tasks, simulation design enables them to be assigned to agents dynamically, and their activation conditions and specific work can be parametrizable by users. The design included different types of tasks to capture specific team needs (see Figure 6):

- *Target-based tasks:* It is a task that can be parametrizable with a given target to follow (usually a mobile target), and can define different behaviors to accomplish this goal.

- *Strategy analyzer:* It is a task representing all the selected strategies (i.e. other tasks) for a given team. These tasks are statically defined by a coach, and they are not attached to any particular player.
- *Temporal strategies:* Usually, players change temporarily their behaviors to participate in a given situation, and then their return to their previous usual behaviors. This feature was implemented with a task structure able to recognize special conditions activating a predefined situation, store agent states, change some agent properties to effectively take part in the situation, and then restore the old states when the situation is finished. These tasks are also separated from players, but involve more dynamic behavior than the static tasks.



Strategy Analyzer. In this situation, the coach wants a player pressing on the ball when the ball is in the defense zone. Thus, the *BallPressionModeTask* task (in *StrategyAnalyzer*) is activated when a ball agent is detected in the selected field, and this task sets to 1 the value of the *defenderPressing* property. There is another task that is activated when the ball is out of the defense zone, turning that property to 0.



Temporal Strategy. In this situation, a defender takes a forward position in a corner. The initial condition (*startCondition*) activating this task (*GoToCornerKickTask*) is a corner event. The temporal behavior comprises modifying the offensive position in the player towards a given location in the rival area. When someone scores or a rival player catches the ball during the play, an ending condition (*endCondition*) occurs and the values in the player are restored to their original values.

Fig. 6. Some examples of strategy tasks in the simulation

- *Predefined plays:* With these tasks, a coach can specify a more abstract play, defined by a target player (generic) and a set of scenes depicting the play. The target player might decide to abort such play because there are better alternatives to follow. The specific mapping of a template play is instantiated during runtime.

Using these facilities, a support to define, apply and change different team strategies during a game was implemented and tested.

4.3.2. Results and lessons learned

Implementing a soccer simulation using a framework (apparently quite distant from that domain) was an interesting challenge. The results obtained were really encouraging, and we present some of them:

- *Concurrency.* Concurrency was experimented in some previous examples, but it became necessary in the soccer simulation. It allows a more real modeling of coordination situations during the game. A single agent is now able to execute some of its tasks in a concurrent mode with other agents.
- *Information repository.* A common repository of meta-level information was used as an alternative to implement direct data exchange between agents. We think that event flow can be reduced in some cases using this technique. Debugging and traceability could be also simplified with this option.
- *Group management.* The original composed-agent derived in a coach/team structure, where the coach has the responsibility of defining suitable strategies for the players. Other envisioned possibility involves more elaborated reasoning in these manager agents.

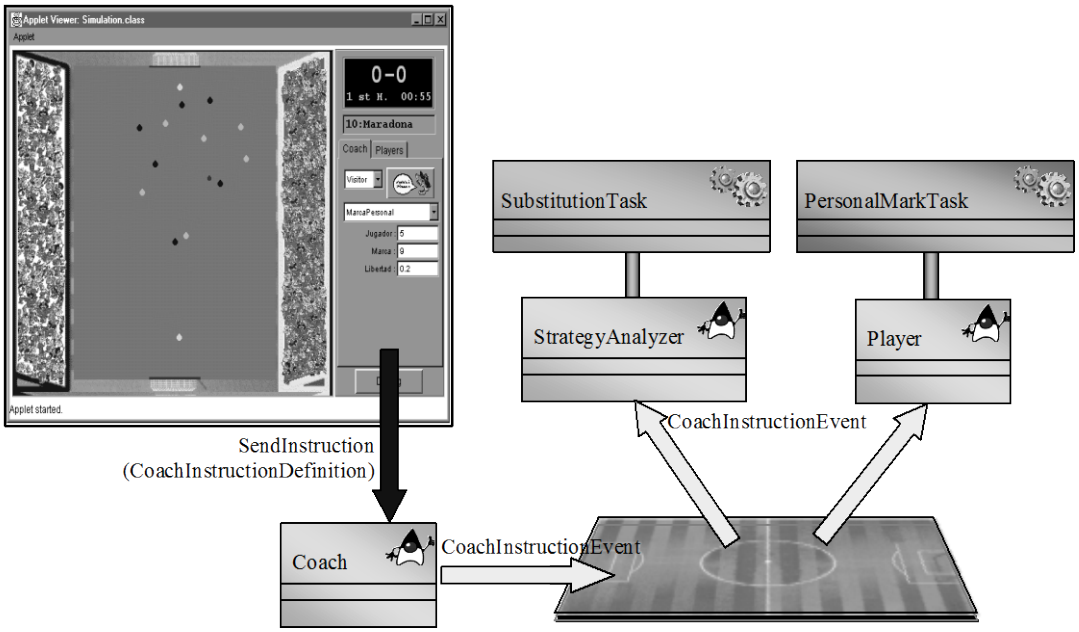


Fig. 7. Using the event mechanism to send coach instructions (via a GUI) to the players

- *More complex tasks.* The development of this example had a particular characteristic: our implementation team did not understand completely how the roles of the tasks worked into the framework. As a consequence, they did not use the framework in the usual way, even though an important part of their work was related to different types of tasks. Strangely, this situation did not produce a negative effect at all. We discovered new possibilities about agent behaviors involving coordination patterns based on those tasks.
- *Coordination.* The soccer simulation required to divide tasks in two categories: basic tasks (following a common task structure inherited from the existent architecture), and more specific tasks related to the roles played by agents in coordination protocols. This idea shifted our conception about agents in the framework, extending the concept of competing tasks. Now, an agent could be defined as a series of roles involving predefined tasks, and each role might be enabled or disabled at runtime as a result of the current collaborations taking place among the agents. Moreover, we believe that certain coordination patterns can be extracted from these multiagent simulations. Following these lines, they could be specified and applied to other problems.

5. FUTURE WORK

The case-studies presented in this paper are only a small part of the applications that can be developed with the Bubble framework. Regardless of this experience, we believe that Bubble's architecture is still in evolution, and it has to provide developers with a well-established set of hooks to build multiagent simulations on top of the framework. The ideas about roles and coordination patterns are emerging, and a more digging work could derive in useful results to define agent organizations in simulation models [Lesser98].

Future work includes settling the existent capabilities, and providing other features such as a visual environment to manipulate the framework and distribution capabilities. On the other hand, we should be careful in this process, because a wide range of built-in features can affect and limit framework adaptability. Problems of tangling code might arise, leading to difficult maintenance and usage of software. A clear separation of concerns should be studied to tackle these issues. When it comes to other application examples, we are planning to test the framework in systems involving workflow and plant control.

6. CONCLUSIONS

The use of object-oriented application frameworks promotes reuse and adaptability, but only if these quality factors are enforced during design. The experience reported in this paper shows how the mechanisms provided by Bubble's architecture (i.e. agents, events, filters, tasks, sensors and uniform decomposition) resulted in an adaptable support. We can remark the following items:

- The containment notion was relaxed to admit other group structures.
- Original tasks formed by combination of basic blocks became more complex (specially with the third example) to support parametric definitions and team strategies.
- A simple hierarchical organization derived into agent groups coordinated by other agents and different roles (played by the agents) according to specific strategies.
- Communication via events was enriched with other mechanisms such as a blackboard approach.
- Other features included separation of geometrical concerns, concurrency issues and visual tools.

Bubble was developed using a multiagent approach based on reactive agents, and it should be seen as an alternative tool of multiagent modeling to simulate complex realities. It is interesting to analyze the evolution of the first basic framework from the early applications to the final ones (See Table 1). On the other hand, framework adaptability implied a wide set of possibilities to configure a given application, but sometimes this situation can produce certain disorientation in the users regarding to what features they should use. We think that the definition of some common programming model to create applications using the framework is also required. A visual environment might help in this task.

Aspect	Fluids	Mobile hosts	Soccer
Structural organization	Strict hierarchical containment. Geometrical considerations.	Groups of agents where containment is not necessary. Separation of geometrical concerns.	Agents belong to a team, coordinated by another agent. Roles.

Tasks	Simple tasks, or combinations of blocks.	Simple tasks, or combinations of blocks.	Parametric tasks. More dynamic structures. Tasks involving roles, applied to teams.
Communication	Events and sensors. The event flow is managed by containers.	Events and sensors. Broadcast and multicast events.	Events and sensors, plus other mechanisms such as a blackboard.

Table. 1. Evolution of the Bubble's architecture

Finally, the results obtained with the framework and its ulterior applications were encouraging from an architectural point of view, and we believe they have enough credits to generate new ideas and future research about organizational aspects (roles and coordination) in multi-agent simulation frameworks. Dealing with adaptability in object-oriented application frameworks is not a minor issue, Bubble may be just a strange case, but in some manner it is pointing out the right way towards engineering adaptability in software systems as a critical factor to face unexpected changes.

Acknowledgments

Thanks to Martín Lahittette and Maximiliano Keen who implemented the soccer simulation using the Bubble's architecture and provided us a valuable feedback about framework usage. Thanks also to Alejandro Clausse for his generous help in physical issues related to fluid flow problems.

REFERENCES

[Demazeau91] Demazeau, Y., and Müller, J. P. eds.: *From reactive to intentional agents*. In Decentralized Artificial Intelligence 2, pp. 3-14. Elsevier/North-Holland, Amsterdam. 1991

[Drogoul92] Drogoul, A., Ferber, J.: *Multi-agent simulation as tool for modeling societies: Application to social differentiation in ant colonies.* Proc. Eur. Workshop Modelling Autonomous Agents Multi-Agent World, 4th,Rome, Italy. 1992

[Perkins94] Perkins, C., Myles, A., and Johnson, D.: *The Internet mobile host protocol (IMHP),* Proceedings INET'94, Annual Conference of the Internet Society. Czech Republic, June 1994

[Fayad96] Fayad, M., and Cline, M.: *Aspects of software adaptability*, Communications of the ACM, 39(10), 58-59. 1996

[Herrero96] Herrero, V., Clausse, A., and Guido-Lavalle, G.: *Geometrical automata for two phase flow simulation, Technical note.* Printed from Nuclear Engineering and Design 163 117-124. 1996

[Moulin96] Moulin, B., and Chaib-Draa, B. : *An Overview of Distributed Artificial Intelligence*. In "Foundations of Distributed Artificial Intelligence", Chapter 1, edited by G.M.P. O'Hare y N.R. Jennings. A Wiley-Interscience Publication. 1996

[Shaw96] Shaw, M., and Garlan, D.: *Software Architecture, perspectives on an emerging discipline*,Chapter 2: Architectural Styles. Published by Prentice-Hall. 1996

[Fayad97] Fayad, M.E. and Schmidt, D.: *Object-Oriented Application Frameworks*. Communications of ACM, Vol. 40, No. 10, pp. 32-38, October 1997

[Johnson97] Johnson, R.: *Frameworks = (components + patterns).* Communications of the ACM, Theme issue on "Object-oriented application frameworks", Mohamed E. Fayad and D. Schmidt (Eds.), 40(10) pp. 39-42. 1997

[Meyer97] Meyer, B.: *Object-Oriented Software Construction*. Second Edition, Prentice-Hall. 1997

[També97] També, M.: *Towards Flexible teamwork*, J. Artificial Intelligence Research (JAIR) 7, 83-124. 1997

[Bass98] Bass, L., Clement, P., and Kazman, R.: *Software Architecture in Practice*, Chapter 6: Unit operations. Published by Addison-Wesley. 1998

[Lesser98] Lesser, V.: *Reflections on the Nature of Multi-Agent Coordination and its Implications for an Agent Architecture*. Appeared in "Autonomous Agents and Multi-Agent Systems", Kluwer Academic Publishers, 1, 89-111. July 1998

[Stone98] Stone, P., Veloso, M., Han, K. and Achim, S.: *CMUnited: A team of robotic soccer agents collaborating in an adversarial environment*, In Hiroaki Kitano, editor, RoboCup-97: The First Robot World Cup Soccer Games and Conferences. Springer Verlag. 1998

[Sycara98] Sycara, K.: *Multiagent Systems.* In AI magazine Volume 19, No. 2. 1998

[Campo99] Campo, M.: *Invited presentation, Our soccer simulation*. 28 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina. 1999

[DiazPace99] Diaz Pace, A., Trilnik, F., Clausse, A. and Campo, M.: *BUBBLE: a framework for simulation of collective processes using reactive agents.* Proceedings SyM'99 (Simulación y Modelística) Pp 50-65, 28 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina. 1999

[Fayad99] Fayad, M., Schmidt, D., and Johnson, R.: *Building Application Frameworks: Object-Oriented Foundations of Framework Design.* Wiley Eds. 1999

[Stone99] Stone, P. and Veloso, M.: *Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork*, Artificial Intelligence,110(2):241—273. 1999