Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 5, May 2013, pg.408 – 416

RESEARCH ARTICLE

Stimulus-Response Behavior of Moving Agents

Yuniel E. Proenza Arias[1], Martín E. Proenza Arias[2]
[1]Universidad de las Ciencias Informáticas, Cuba
[2]Desoft Granma, Cuba

[1] *yproenza@uci.cul;* [2] *mproenza@grm.desoft.cu*

Abstract— This paper considers the fact of designing the stimulus-response behavior of spatial moving agents as a Command pattern-based approach. By decoupling the way agents receive the information from the actions they perform to react to that information, we find a design that is scalable and adaptable, as well as beneficial at the time of making improvements (i.e. for performance). We discuss such advantages and give some theoretical considerations and guidelines on how each component should be used and the relationships established among them. The design may be treated as part of the kernel of a more sophisticated system, like a decision engine, but it does not cope with it by itself. We present a case study where the kernel of the design is evolved and applied to a particular problem.

Key Terms: - Autonomous Agents; Command Pattern; Stimulus-Response Behavior

I. INTRODUCTION

The behavior of autonomous moving objects (agents from now on) generally involves the interaction between them and the environment. This interaction normally includes a large series of steps, like getting information from the world, filtering the information, reasoning and making decisions and responding. There are many modalities of responses an agent can perform, like avoiding obstacles, staying on the path or following certain object; the response might also be the last part of a long process of thinking and computing the information.

While the design of such systems may differ, the part of getting information from the world and transmitting it to the unit capable of dealing with it remains present in every system of this type because it is such a basic concept that it cannot be removed.

The representation of the senses of agents has evolved to sophisticated models of sensory perception. We can classify an agent's perception into different modalities as in [1]: sight, touch, hearing, smell and taste. Each of them can be represented as a physical entity considering that these senses have a finite extension in space (sight cone, body, ears, nose and tongue). Senses are activated by stimuli, which can be represented physically as well. Hence, the interaction between agents and the environment can be modeled as the interaction of physical entities (senses and stimuli). The objects an agent can sense define what is called the local world state, that is, the properties of the world around the agent that change. Other global states like the time of the day are not "sensed" by agents and are out of the extent of this paper; although they may interfere in the final agent's reaction (i.e. the agent might be tired).

Giving spatial objects the capability to react to environmental stimulus may be approached using different methods: *Steering Behaviors* [2] [3] and *Potential Field-based Movement* [4] are some of the most popular; a general idea of many of them is given in [5] [6]. While they might be pretty different in their essence (they provide different interfaces), it is sometimes needed to incorporate all of them into a system because each provides useful features. So, creating a full size behavioral model for a specific agent involves bringing different techniques together in just the right way.

Another aspect of these systems that is a major concern is the diversity and constant improvement of techniques for making spatial queries taking into account that this operation can happen to be the most critical computational cost in many applications. New and more sophisticated methods are devised, many of them supported on algorithms over data structures based on the decomposition of space. The structure used generally depends on the particular problem. For instance in [7] a balanced-box decomposition tree (BBD-tree) is used to solve the problem of finding the nearest neighbor of objects with complex representation; in [8] a 3D lattice of box-shaped voxels is used to accelerate agents' neighbors search in a crowd simulation while in [9] the same structure is used to take advantage of hardware architecture. Another different structure has been used in [10] and others are described in [11].

When building an engine of Artificial Intelligence behaviors we should think of having a fixed structure that works as the core, so that it supports other components to be built upon and which can be easily reused, applied and combined in interesting ways. It is this adaptable core what we are concerned of.

We emphasize two main aspects throughout this paper: extending perception systems (way of getting data) and extending response systems (way of reacting).

II. MOTIVATIONS

The applicability domain of the design we discuss further can be described with the following example. Here, we refer to agents as any entity in a client application that somehow uses the system's output.

Consider that we have certain agents which we must incorporate the capability to effectively perform actions to respond to the environment (i.e. avoid obstacles, flock with other agents). We are not aware of the exact capabilities we must give to them or the methods to be used to cope with such capabilities, as well as it is not specified the interface they might provide to hold them. The objects they must react to are not specified neither. Different agents might have access to different information or can share the same. The problem of storing and acquiring data from the environment is presumably the one that affects performance the most.

We must design a kernel (system) where to build upon that adjusts to these requirements. Figure 1 gives us a global view of how the system should look like:

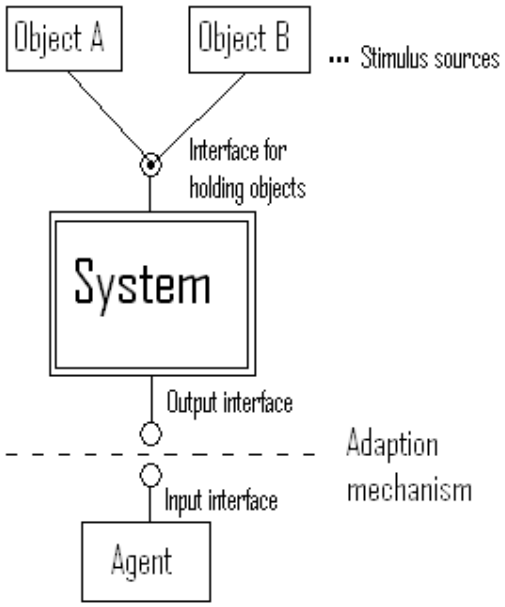

Figure 1. System global view

Here, objects A and B represent stimulus sources, a more general concept that may include not only physical objects like a rock or a car, but also abstract ones like sound or wind. We refer to objects and stimulus sources interchangeably.

Preliminary Analysis

From the previous description we can extract the scope of the problem as being the one described below. We provide guidelines for the solution of each issue, which will lead us to the final design:

Issue: New types of data (stimulus sources) might be added for the agents to react to, so new ways of storing and acquiring such data might be needed. The same data might be accessed in different ways.

Guideline: Separate the way of storing data from the way of accessing it. By defining info sources and sensors separately we could define in each the responsibility of storing data and the responsibility of acquiring it respectively.

Issue: Performance improvements might be placed where the storage and access to the information is carried out.

Guideline: Have the way of storing data and the way of accessing it work together and support each other. The way of retrieving data from the info source greatly depends on the particular structure (way of organizing information) of the info source (i.e. books in a library are organized by topic and authors in bookcases, while crops in a storage facility are organized by type and size of the package in enormous stacks).

Info sources should not have a built-in structure because that depends on the types of objects it holds (i.e. in the case of books and crops, the way of storing each depends on their characteristics and shapes).

Because there may be no dividing line between the world interface and the way data on it is accessed (i.e. a unique way of querying the world), it should be a responsibility of each type of info source to define its own way of accepting queries.

On the other hand, there must be a correspondence between the shape of the sensor and the way of querying the world, considering that the interaction of the sensor with the objects in the info source is purely physical. Figure 2 summarizes the collaboration of info sources and sensors.

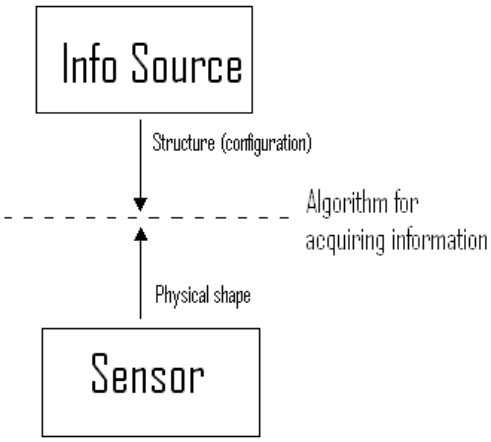

Figure 2. Collaboration between Info Sources and Sensors

Here, info sources are a means of representing each part of the world differently, so that equipments access data in the right and most efficient way.

Issue: Information sources might vary from agent to agent.

Guideline: Have each agent bound to its particular information sources. This can be accomplished through binding each sensor to a specific info source (i.e. by composition).

Issue: The agents' input interface is not known.

Guideline: Have the system totally decoupled from the agents' input interface. Have a way of configuring the system's output to suit the agents' interface. This configuration should be provided by the user since there is no way of anticipating it. We should be capable to adapt (by building on top of the kernel) the system's output to the agents' interface and not vice versa.

Issue: The capabilities to incorporate to the agents might gradually evolve and might be quite diverse.

Guideline: Have a way of implementing different response mechanisms without them having to conform to a specific interface and still be pluggable into the system. We could plug several mechanisms to each sensor via a signal. Each signal would activate a mechanism, which would then execute actions in the agents allowing them to react to the information collected, binding agents to the system.

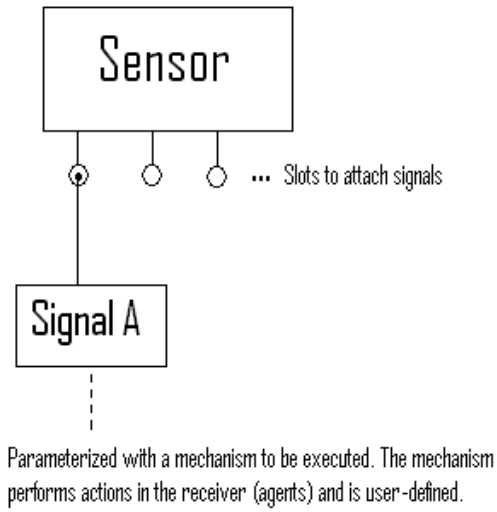

Figure 3. Binding mechanisms to the System

The overall structure of the system is the following:

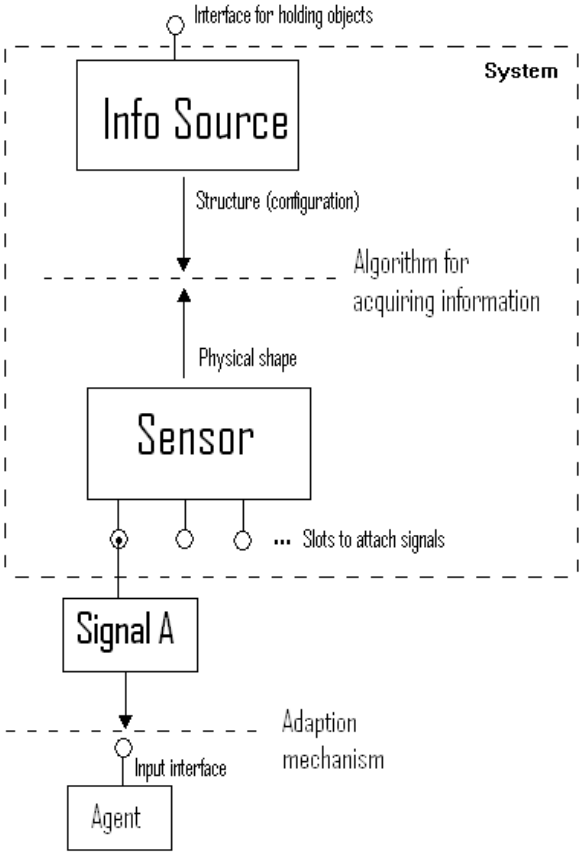

Figure 4. System overall structure

III. SOLUTION

We employ *Command* pattern to decouple the way agents receive the information from the actions they perform to react to that information. The roles are the following: *Agent* is the *Client*, *Sensor* is the *Invoker*, *Signal* is the *Command Interface*, a *Concrete Signal* is a *Concrete Command* and a *Concrete Mechanism* is a *Receiver*. See [12] for specs of *Command* pattern.

In the assumption that agents do not have means of getting information from its environment, we provide such capability with the introduction of sensors. Sensors not only provide a way to acquire information from the world, but we may also create different types of sensors by subclassing, and so, get different ways to access data in the world.

Info Source is an abstract class that works as a utility and has the responsibility of storing objects. *Info Source* does not have a built-in structure; classes deriving from it must define their own in order to answer locality queries as fast as possible. A huge part of performance is due to the efficiency reached with the structure defined in each info source.

We also provide an update() function in *Info Source* and *Sensor* so that information in both remain refreshed. The final design is represented in figure 5:

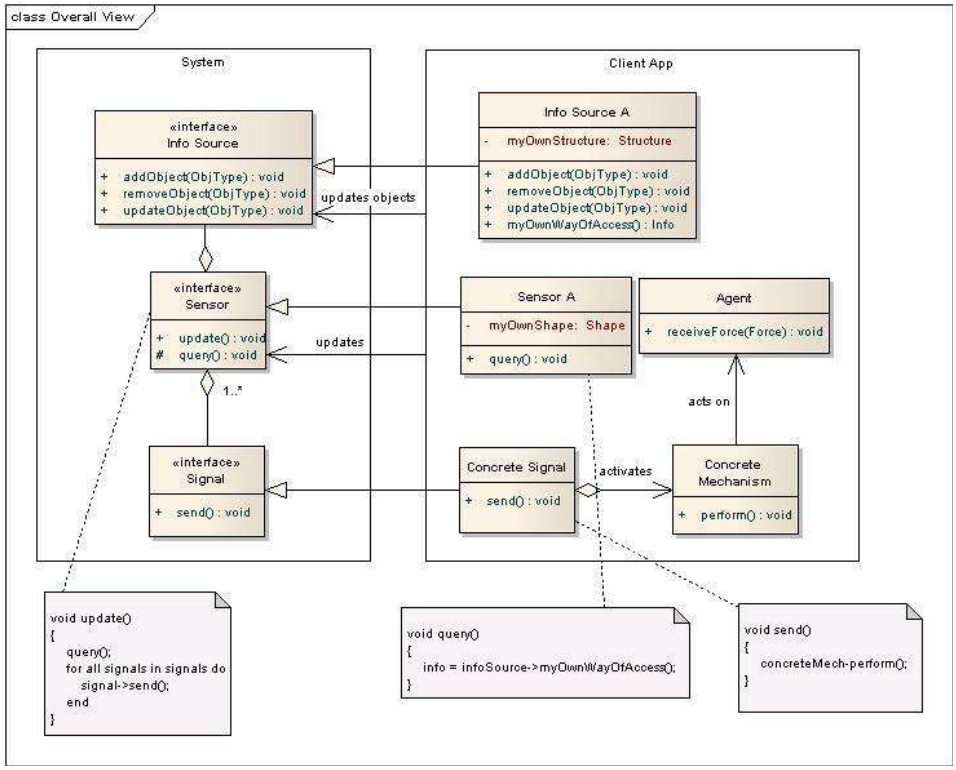

Figure 5. The System and the Application using it

The figure clearly separates the kernel (system) design and what a client application might implement. Each info source declares its own way of accessing data on it and query() function in each sensor is in charge of acquiring that data by calling the appropriate function in a particular info source. *Template Method* pattern is used to establish a standard behavior of the sensors through update() method.

IV. DISCUSSION

Even when the kernel has different features, it is its *Command* pattern-based design what gives it the strength of being adaptable and scalable. By building above this kernel we may achieve from simple stimulus-response agents (agents that do not think, just act) to more complicated and refined agents that think and decide what to do and when, by wrapping this kernel with a more sophisticated decision making system.

Notice that as we extend *Info Source*, *Sensor* and *Signal*, the interface of the system remains the same as in figure 4. Moreover, it has been added an interface for updating the system. The following are implications of the design:

- The interactions between agents and the environment are modeled in terms of physical interactions between sensors and the objects in the info sources. Hence, *all objects must be treated as physical entities* (i.e. there must be a way of computing its extension in space).
- Sensors by themselves are a means of detecting whether an agent can physically sense certain information. *We can distinguish the information an agent reaches for that it does not*.
- Since mechanisms do not have to share a common interface, *there is an infinite possibility of addition of new mechanisms*, as long as there is a well-known protocol (data filtering, data conversion, etc) for communicating with them. This protocol should be encapsulated inside the signal objects.

- Sensors all share a common interface, which *can be used to extend new perception systems and still get from it the behavior required to perform well within the full design.*
- *Sensors characteristics may be varied on execution time without affecting any logic in the agents*. For instance, we may have an agent that is constantly looking around while walking; we just have to make a sensor rotate independently of the agent.
- Having the perception systems being decoupled from agents brings two main advantages: (1) *adding new perception systems or modifying the existing ones do not imply any change in Agent class* and (2) *we may save unnecessary calculations in the agents' logic in case no relevant data is found by the sensors*, since we can make agents' logic remain unexecuted (i.e. by deactivating mechanisms).
- Since mechanisms are user-implemented, *they control the adaption of the output interface of the system to the input interface of agents*.
- We reach a comprehensible structure where *responsibilities are very clear so the places for improvements are localized easily*. For instance, locality searches are a responsibility of info sources, while the capability to react to objects in the world is encapsulated inside a set of mechanisms; both can be improved by modifying or adding new info sources and mechanisms respectively.
- By letting sensors do queries on the world (and not have the world announce its data to the sensors) *we may gain in performance by letting each sensor adapt to the info sources' special requirements for effectively querying it*. The general data access performance would be as fast as the slower info source-sensor's data acquiring algorithm.
- When equipping (attaching sensors to) an agent, we bind it to specific info sources, so that *different agents might gain access to different data*. *We can discriminate between the data an agent has access to and the data it does not*.
- By implementing a new triplet *Sensor-Signal-Mechanism we are able to obtain a whole new branch of capabilities for the agents* (new way to obtain data and new way to react to that data).

Sometimes a mechanism needs data that is not contained in the info sources (i.e. an obstacle collision detection mechanism might need information of the shape of the sensor). This brings up two main concerns depending on the implementation: (1) the information must be passed through at some place, either by delegating that responsibility to the sensor in the implementation of query() function or by delegating it to the signal or the mechanism that is going to use it; and (2) type safety issues might appear. See [12] for further implications, benefits and liabilities of *Command* pattern.

Further notes on implementation

The flexibility of this design might be improved or it can be modified in order to match specific rules in the applications it shall be used. Here are some implementation tips that may be considered:

- If the interaction between the info source and the sensors is complex, it may be encapsulated in an isolated object using *Mediator* pattern to encapsulate the way of querying the info source. This would allow a loose coupling between info sources and sensors by making them act as *colleagues* (see [12] for specs on *Mediator* pattern). Consequently, the mediator object would implement the algorithm for acquiring information (the boundary dotted line in figure 2)
- We may have the sensors have a state variable, which indicates whether it is active or not, and so control when to use a sensor and execute the performance-spoiler process of getting information. Another approach would be to have a list of sensors to be updated and insert or remove for convenience. A scheduling mechanism might sustain this process.
- Instead of simple mechanisms we can plug in a whole system of filtering to the kernel. The kernel might be a thin layer underneath a sophisticated decision, planning, or learning system.
- It could be used template features, depending on the programming language, to enhance and ensure the integrity of the system. For instance, each info source could be parameterized with the type of object it holds, as well as signals could be parameterized with the type of mechanism it activates and the mechanism be passed to the signal at construction time.
- We can implement a system where signals are computed independently of the time they were sent. We may have signals connecting mechanisms that might work at different times (i.e. with a commands processor/scheduler).
- We may implement the design as a parallel processing system to gain in performance, where each signal is "sent through" a different thread.
- Because the world is a composition of different info sources, each having a different way of querying it, it is reasonable that we might want to have a single central world interface system where to find and place changes.

V. CASE STUDY

Problem: we have agents that expect forces from the system in vector form to accumulate them, summarize the action of each (see [13] for details on summing methods) and finally apply a predefined locomotion system that uses the resulting force. Agents do not have a way to get information from the world or even know what kind of objects they might have to react to.

We want to give these agents the capability to flock with other agents or avoid them (if of the same or different kind). We also want them to avoid polygonal and circular obstacles. We want to use *Steering Behaviors* to perform flocking and avoiding polygonal obstacles (which includes collision detection) and *Potential Field-based Movement* to avoid circular obstacles. Plus, good performance is a requirement.

Solution: In this case, we have two kinds of objects: obstacles (polygonal and circular) and agents.

We decide to separate the way of storing each kind of objects. We will use a grid-shaped partition (GSP) (a type of spatial partition; see [13] for specs) for organizing the storage of agents (in the hope of improving performance in locality queries) and a simple storage facility (SSF) for obstacles, which returns the full list of obstacles in the world every time it is queried. We model the agent's vision as a sensor as represented in figure 6. This sensor is used to find agents within the view range. Its info source is the GSP. It is attached the signals connected to the mechanisms to flock.

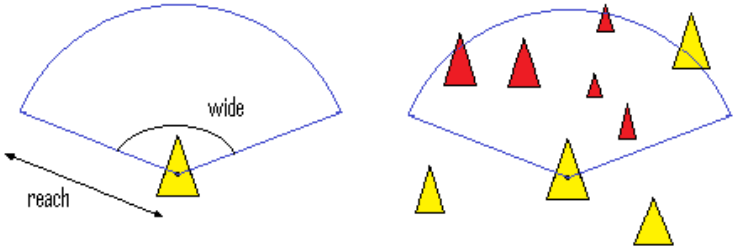

Figure 6. View Range sensor

Polygonal obstacle avoidance is modeled by using a Box-Shaped sensor as represented in figure 7. This sensor is used to detect collisions with obstacles (may include a collision detection algorithm). Its info source is the SSF. It is slotted in the signals connected to the mechanisms to avoid polygonal obstacles.

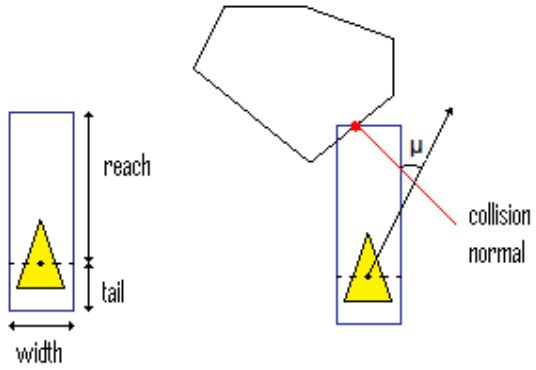

Figure 7. Box-saped sensor.

Circular obstacle avoidance is modeled by using a point-shaped sensor as represented in figure 8. This sensor detects the potential field's intensity of circular obstacles. Its info source is the SSF. The circular obstacles avoidance signal is connected to it.

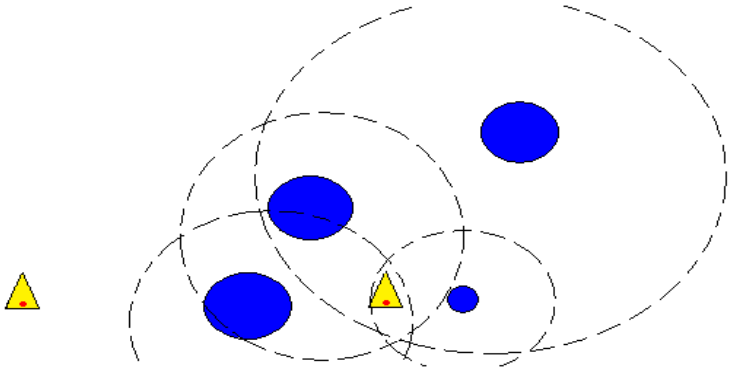

Figure 8. Point-shaped sensor.

The interaction of the classes ends up being the following:

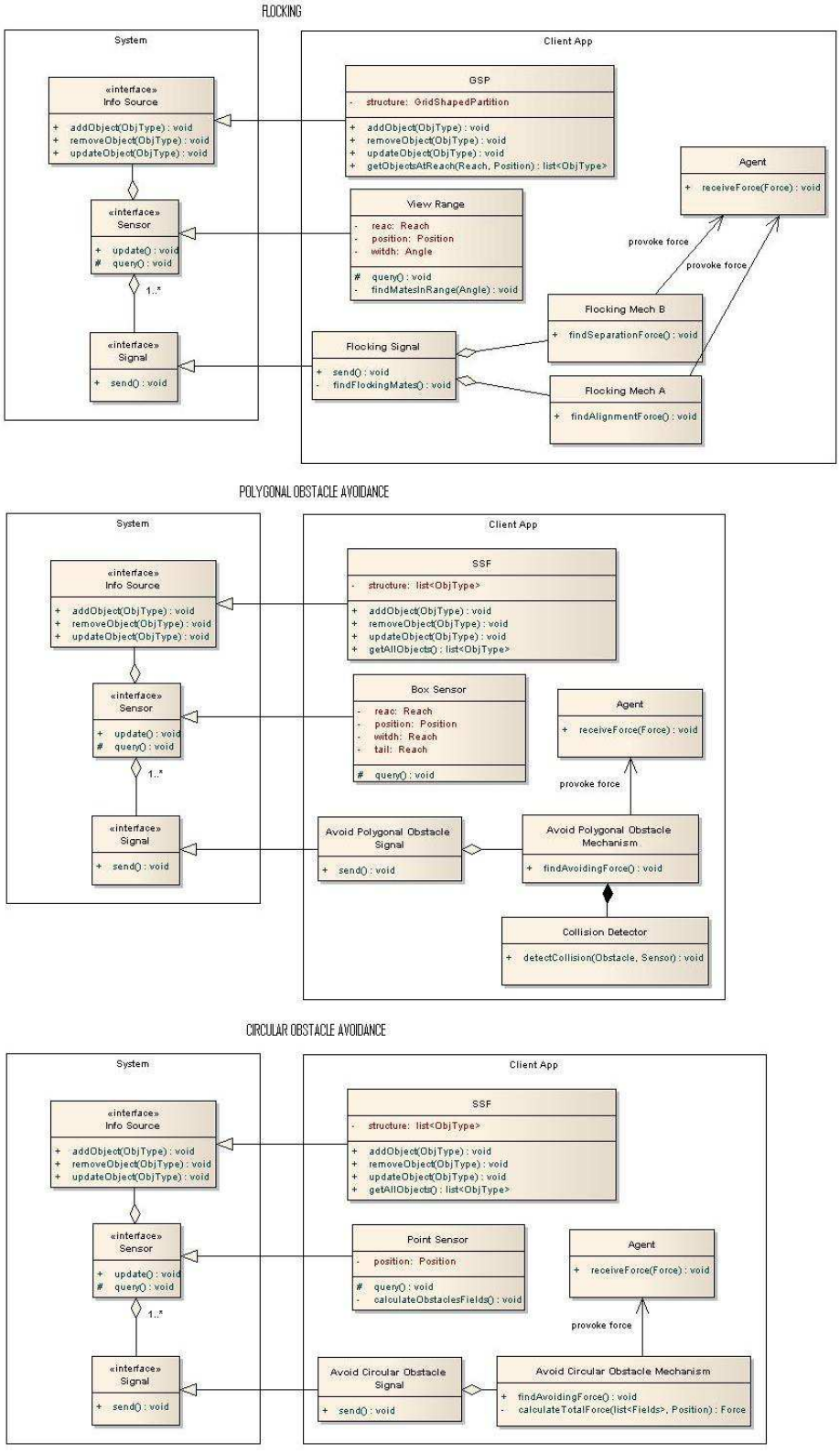

Figure 9. Case Study: interaction of classes.

Highlights of the case study

These are some highlights of the case study:

- Agents themselves constitute a type of object (stimulus source) and so must be stored in an info source.
- The mechanism for querying the world depends on both the sensor's shape and the info source's structure. In all cases, both integrate with each other (i.e. GSP-View Range and SSF-Box Sensor/Point Sensor).
- A sensor or a signal's responsibility might be enhanced. In this case, we give the flocking signal the responsibility to find the flocking mates of the agent and the view range the responsibility of finding mates according to its aperture width. In general, filters can be applied in any place from sensors to the final destination, in this case, agents).
- The mechanism to avoid polygonal obstacles receives different data types. The info sent to this mechanism must then include different types of objects, in this case, the equipment and the list of obstacles, both needed to test collisions. The responsibility of attaching extra info depends on the design for passing it from sensors to the final target (i.e. we could pass the info in send() function or buffer it in memory and have it referenced).
- The GSP must be aware of the representation of agents (i.e. to get their position) in order to answer locality queries. In general, info sources must have at least a constrained access to the objects it holds (i.e. access to a part of its interface).
- A signal might be connected to more than one mechanism. In flocking signal we take advantage of the fact that all flocking mechanisms share the same data (the list of agents within the view range).

VI. CONCLUSIONS

While the design discussed here does not provide a full-featured set of services, it does break up the model into different entities capable of coping with the full problem from getting information from the world to defining how to react to this information, and specifies the responsibility of each component, letting concrete implementation be under user's choice.

We can build above the kernel and adapt it to our own needs by extending *Info Source*, *Sensor* and *Signal* classes. The system may grow to the infinite due to that it is always possible to adapt the system's output interface to any client application's input interface. We demonstrate that *Command* pattern can help very much in obtaining flexible designs for systems that are liable to be extended with the addition of new algorithms without them having to conform to a specific interface.

REFERENCES

[1] I. Millington. Artificial Intelligence for Games, Morgan Kauffman Publishers, 2006.
[2] Craig Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model. In Proc. of SIGGRAPH'87, 1987.
[3] Craig Reynolds. Steering Behaviors for Autonomous Characters. In Proc. of SIGGRAPH'99, 1999.
[4] M. Mohan, D. Busquets, R. López de Mántaras, and C. Sierra. Integrating a Potential Field Based Pilot Into a Multiagent Navigation Arquitecture For Autonomous Robots, 2003.
[5] David Bourg and Glenn Seeman. AI for Game Developers, O'Reilly, 2004.
[6] Stuart Russel and Peter Norvig. Artificial Intelligence: A Modern Approach, Prentice Hall, 2002.
[7] S. Arya, D. M. Mount, R. Silverman and A. Y. Wu. An Optimal Algorithm for Approximate Neighbor Searching in Fixed Dimensions. Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, 1994.
[8] Craig Reynolds. Interaction with Groups of Autonomous Characters. Games Developers Conference'00, 2000.
[9] Craig Reynolds. Big Fast Crowds on PS3. Sandbox'06 (an ACM video games symposium), Boston, Massachusetts, 2006.
[10] J. Derryberry, D. Sheehy, D. Sleator and M. Woo. Achieving Spatial Adaptivity while Finding Approximate Nearest Neighbors. CCCG'08, Motréal, Québec, 2008.
[11] Jon Louis Bentley. A Survey of Techniques for Fixed Radius Neighbor Searching. Stanford Linear Accelerator Center, Stanford University, 1985.
[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Longman Inc., 1998.
[13] Mat Buckland. Programming Game AI by Example, Wordware Publishing, 2005.

