# Caribbean/*Q*: A Massively Multi-Agent Platform with Scenario Description Language

Yuu Nakajima
Kyoto University,
Dept. of Social Informatics
nkjm@ai.soc.i.kyoto-u.ac.jp

Hironori Shiina
Kyoto University,
Dept. of Social Informatics
shiina@ai.soc.i.kyoto-u.ac.jp

Shohei Yamane
Kyoto University,
Dept. of Social Informatics
yamane@ai.soc.i.kyoto-u.ac.jp

Hirofumi Yamaki
Kyoto University,
Dept. of Social Informatics
yamaki@i.kyoto-u.ac.jp

Toru Ishida
Kyoto University,
Dept. of Social Informatics
ishida@i.kyoto-u.ac.jp

## Abstract

*Making a truly useful massively multi-agent system is difficult since the actions of the full ensemble of agents cannot be controlled by designing just one agent. It is critical to control all the agents by using protocols that describe the interaction of agents and the environment in a top-down approach. We introduce a system that uses interaction protocol descriptions and has the capability of controlling hundreds of thousands of agents. This makes it feasible to realize a mega-scale navigation system that can assist the inhabitants of a small city. In developing a massively multi-agent system, protocol design and agent development need to be separated to allow specialists to work in concert with one another while honing the different technologies. As a platform for mega-scale navigation system, we devise an architecture for multiagent platforms where the execution of agents scenario and the implementation of agents are explicitly separated. This paper also gives the evaluation and an application example using the platform.*

## 1. Introduction

The spread of mobile terminals like cellular phones and PDAs, and positioning systems like GPS, will realize a ubiquitous environment for city-dwellers. Using this environment, we can build a large-scale navigation system (*mega-scale navigation system*) like traffic control or evacuation navigation for any particular city [4]. Current systems simply broadcast the same instructions over a large area, but what is needed is a system that can provide individualized instructions to each person.

Our approach to provide personal navigation is to build a multi-agent system that assigns one Guiding agent to each human. In this system, an agent can provide personalized navigation instructions considering the human's characteristics, city-supplied evacuation targets, and the surrounding environment.

Making a massively multi-agent system work properly is difficult if only single agent is designed. Therefore, it becomes important to control agents by describing interaction protocols[1] predicting agent interaction in a top-down scenario. In this paper, we describe the construction of a system that uses protocol descriptions to control hundreds of thousands of agents in order to realize a mega-scale navigation system targeting city-scale crowds.

To realize a massively multi-agent system platform, we address the following three issues.

i. **Separation of Protocol Design and Agent Development**
In developing a mega-scale navigation system, experts of the intended domain (ex. traffic or protection against disasters) will design the agent interaction protocols while computer experts will develop the agent system. If the agent platform forces the agent system developers to integrate agent internal models with protocol descriptions, the former must be significantly revised, which is very expensive, if the protocol descriptions are changed. This shows that any truly practical development environment must separate protocol descriptions from the agent internal models.

ii. **Dynamic Protocol Switching**

[1] In this paper, "Protocol" refers to the interactions permitted between agents and the external world (other agents and the environment).

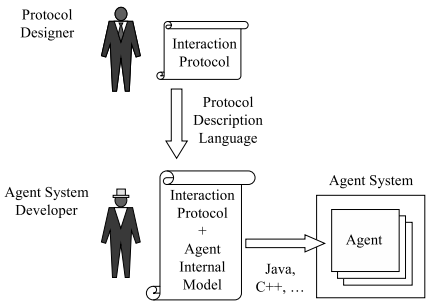


**Figure 1. Both Protocol and Internal Model are Implemented in Each Agent**

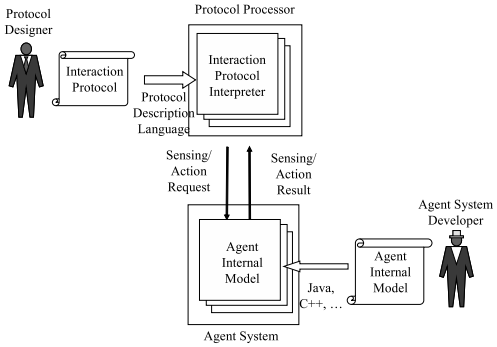


**Figure 2. External Protocol Interpreter Controls Agent System**

In large-scale social systems, each agent faces a variety of situations. A single protocol description to deal with all such situations may become large and complex. Instead, our architecture allows experimenters to dynamically switch protocol descriptions given to agents corresponding to the changing situations.

iii. **Scalability**
Most of existing protocol processing systems and agent systems are not designed with the management of a large number of agents in mind. To manage large-scale social systems, systems have to control a large number of agents that model human behaviors. We achieve the scalability by applying large-scale agent server which is recently developed and works on event driven object models.

## 2. Architecture

There are two possible types for the mechanism to control agents by giving designed protocols. One of them is the one shown in Figure 1, where protocol description and agent internal model are implemented together into an agent. The other is shown in Figure 2, where an external protocol processing system controls agent internal model.

In the approach shown in Figure 1, the developer of the agent system implements an agent by integrating the protocol description, which is given in an inexecutable language such as AgentUML[7], and the agent internal model. In this method where both the protocol description and the agent internal model are implemented in a single agent, the agent implementer has to absorb the knowledge of domain experts first, and then reflects their ideas to agent implementation, which is not efficient. Also, it is hard to switch the protocol according to the changing situations during the operation.

In contrast, the approach shown in Figure 2, the protocol description is given in an executable protocol description language, and an external protocol interpreter interprets it and controls the agent internal model. In this approach, domain experts can directly design protocols without considering the internal implementation of agents. Thus, domain experts and agent implementerscan independently develop a multiagent system.

In this research, we propose an architecture shown in Figure 3 that extends the one given Figure 2 by implementing both protocol interpreters and agent internal models on a large-scale agent server to achieve scalability. A large-scale agent server can manage hundred-thousands of agents by keeping agents as objects and by allocating threads to those objects appropriately. As an example of such large-scale agent servers, we describe Caribbean[8] in the following section.

Since the protocol description and the agent development are separated in this approach as in Figure 2, protocol designers can change protocols without knowing the detail of agent implementation. The protocol interpreter requests the execution of sensing and actions in the protocol given to agents and receives the result, which enables the dynamic switching of protocols given to agents.

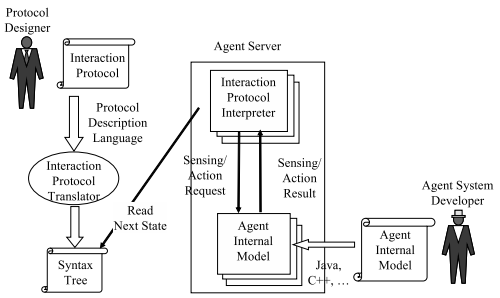


**Figure 3. Protocol Interpreter on Agent System Controls Agent**

MACE3J[1], MadKit[2], Robocup Rescue[5] are also large scale multiagent platforms. MACE3J is a general multi agent platform that realizes scalability because it can perform in distributed environment. Madkit is aimed at integrating a specific social model as a foundation for system design. Robocup Rescue is a platform for the simulation of large scale disaster relief. The platform gets efficiency by dedicates to that.

In these platforms, protocol description and agent development are not separated explicitly. In contrast, our architecture is aimed at realizing the separation of the protocol design and agent the development, which enables the experts of different domains to cooperatively and efficiently develop large-scale multiagent system.

## 3. Fundamental Technologies

We have combined a scenario description language *Q*[3] and a large-scale agent server Caribbean to build a platform for large-scale multiagent system. Below, we describe the two technologies.

### 3.1. Scenario Description Language *Q*

*Q* is an interaction design language that describes how an agent should behave and interact with its environment including humans and other agents. For details see [3]. In modeling human actions, it has been shown that the *Q* approach, describing the interaction protocol as a scenario, is more effective than alternative agent description methods that simply describe the appearance of a human being [6].

The features of the *Q* are summarized as follows.

- **Cues and Actions**
  An event that triggers interaction is called a cue. Cues are used to request agents to observe their environment. A cue has no impact on the external world. Cues keep waiting for the event specified until the observation is completed successfully. Actions, on the other hand, are used to request agents to change their environment. Cue descriptions begin with "?" while action descriptions begin with "!".
- **Scenarios**
  Guarded commands are introduced for the case wherein we need to observe multiple cues in parallel. A guarded command combines cues and actions. After one of the cues becomes true, the corresponding action is performed. A scenario is used for describing state transitions, where each state is defined as a guarded command.
- **Agents**
  An agent is defined by a scenario that specifies what the agent is to do.

### 3.2. Caribbean Agent Server

Caribbean is a large-scale agent server implemented in Java language.

Caribbean manages agents as objects. There are two types of objects in Caribbean, service objects and event driven objects. Objects in Caribbean communicate with each other using Caribbean messaging facility. Service objects can be run at any time and are used for implementing such modules as databases with common information which are frequently accessed. In contrast, event driven objects runs only when they receive messages from other objects. Caribbean scheduler allocates threads to event driven objects based on messages. Usual modules in a system on Caribbean are implemented as this type of objects.

According to our preliminary experiments, if threads are allocated to all the objects to run them concurrently, only up to one thousand objects can be run. Instead, Caribbean enables executing objects of large number by adequately selecting event driven objects to be allocated threads to.

Caribbean limits the number of objects in the memory and controls the consumption of the memory, by swapping objects between the memory the auxiliary store. When the number of objects on memory exceeds a limit, Caribbean moves objects that are not processing messages to the auxiliary store. When objects in the auxiliary store receive messages from other objects, Caribbean swaps them into the memory to process the messages. By performing these swapping efficiently, Caribbean manages a large number of agents that cannot be stored in the system memory at once.

## 4. Implementation

### 4.1. Structure of Caribbean/*Q*

By applying the proposed architecture, we build a scalable multiagent platform that realizes the separation of protocol design and agent development and the dynamic switching of scenarios. We developed a large-scale multiagent platform, Caribbean/*Q*, by combining scenario description language *Q* and large-scale agent server Caribbean based of the proposed architecture. Figure 4 depicts the outline of the system. A *Q* scenario describes an interaction protocol between an agent and the outer world.

The conventional processor of *Q* language, which is implemented in Scheme, cannot control enough agents to realize massive navigation. Therefore, it is necessary to develop the new processor of *Q* language which is available on the agent server Caribbean.

*Q* language is an extension of Scheme, and a *Q* scenario has been interpreted by the processor implemented in Scheme. In order to execute *Q* scenarios on Caribbean, which is implemented in Java, the approach is translating

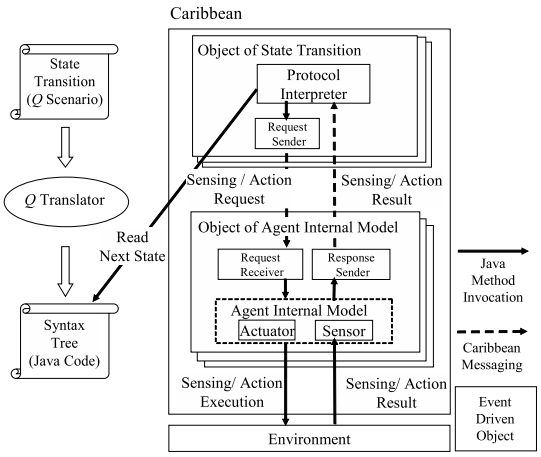


**Figure 4. Overview of Caribbean/*Q***

*Q* scenarios into data structure of Java. This approach gets it easy to handle scenarios on the agent server, and realizes quick execution of scenarios. The translator which translates a *Q* scenario into a syntax tree object in Java is implemented in Scheme. This translation can be realized by parsing a *Q* scenario because syntax of Scheme, which is *Q*'s mother language, is similar to data structure.

In Caribbean/*Q*, the *Q* translator takes a *Q* scenario as input, and converts it to a syntax tree that is read by the state machine object in Caribbean. The state machine executes the converted syntax tree stepwise, by which the protocol given in *Q* is executed. The scalability of Caribbean is thus exploited by importing the *Q* processing system as event driven object in Caribbean.

### 4.2. Execution of Scenario

Since the conventional processor of *Q* language, which is implemented in Scheme, allocates one thread to one scenario interpretation continuously, the number of controlled agents is limited. Therefore, it is impossible to control agents on an agent server, which are much more than threads, with this processor. The proposed method in this research is to utilize event-driven mechanism of the agent server for scenario processing. This method realizes control of many agents on the agent server with scenarios.

Both the protocol interpreter and the agent internal models are implemented as event driven objects in Caribbean. Each agent internal model object has one corresponding state machine object. When switching scenarios, a new state machine object that corresponds to the new scenario is generated and is allocated to the agent.

When the request for the execution of a scenario is given to a state machine object, message exchanges begin between the object and the corresponding agent internal model object. First, the state machine object sends a request as a Caribbean message for the execution of cues or actions to the agent internal model object as a Caribbean message. Then, the agent internal model object executes the indicated cues or actions against the environment, and sends a Caribbean message to notify the state machine object of the result. Finally, the state machine object that receives the result reads the syntax tree, converted by *Q* translator, and makes a transition to the next state. By iterating this process, the given scenario is executed.

## 5. Evaluation

In this section, the performance of Caribbean/*Q* system is evaluated. We compare the performance of the original Caribbean system and that of the Caribbean/*Q* system to evaluate the trade off between the two merits of Caribbean/*Q* (the separation of protocol description and agent development, and the dynamic switching of protocols) and system performance. Also, by comparing Caribbean/*Q* and an implementation where the original *Q* system is externally attached to control Caribbean, we validate the improvement in scalability. The computer used in the following experiment has Xeon 3.06GHz dual processors and 4GB memory, which is enough to keep all the Caribbean objects on memory.

To test the performance that Caribbean/*Q* allocates scenarios to agents, the following simple scenarios [2] with simple cues and actions are used.

```
(defscenario scenario ()
  (scene1
   ((?receive) (!send) (go scene1))))
```

In this experiment, action counters are used to confirm that all the agents execute an action before they go to the next states, in order to guarantee that each agent executes the uniform number of cues and actions and to avoid situations where only a small set of agents run.

The chart in Figure 5 shows the relationship between the number of agents and the processing time for the agents to execute 1,000,000 actions.

From Figure 5, the performance of Caribbean/*Q* is approximately 1/4 of that of original Caribbean. This is because one action of an agent in original Caribbean corresponds to one Caribbean message and that in Caribbean/*Q* corresponds to four messages; the request for the observa-

[2] In complex scenarios, the number of states and the number of parallel observed cues increases. The increase in the number of states does not affect the throughput, since a state transition corresponds to a single edge in the syntax tree. The increase in the number of parallel observed cues does not affect the performance either, since it only increases the number of patterns that shows the names of cues returned from agent internal model objects.

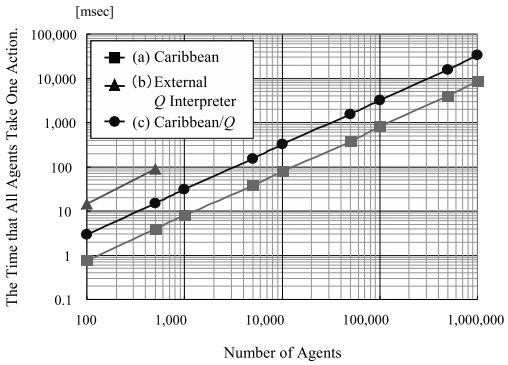


**Figure 5. Evaluation Result of Platform**

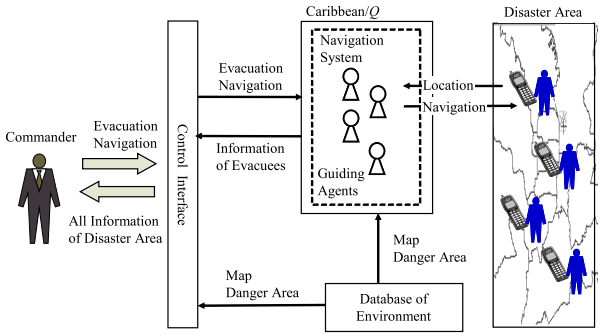


**Figure 6. Overview of Large-Scale Evacuation Navigation System**

tion of a cue its result, the request for the execution of an action, and its result. [3]

The original Caribbean system requires that the data and the functions of an agent are implemented to a single event driven object. In contrast, the implementation of an agent in Caribbean/*Q* is divided into two objects, a state machine object and an agent internal model object, to separate protocol description and agent internal model and to switch protocols dynamically. This demonstrates that there is a trade-off between the two merits in developing multiagent systems and the performance.

As shown in Figure 5, the management of more than thousand agents failed in the implementation where the original *Q* interpreter is just attached externally to the original Caribbean system. In contrast, Caribbean/*Q* successfully managed 1,000,000 agents. The increase in the number of agents does not affect the time to process an action, which means the time to process the whole system is proportional only to the cues and the actions executed.

## 6. Application Example

We produced a large-scale evacuation navigation system for confirming the effectiveness of Caribbean/*Q*. The system commander assigns an evacuation destination and evacuation direction through the control interface shown in Fig. 7. The Commander issues high level instructions to the Guiding agents using a map and the Guiding agents assigned to the evacuees on a one-to-one basis provide individual navigation instructions. Guiding agents give information to evacuees via GPS-capable cellular phones.

In a guidance system which uses ubiquitous information infrastructure on a city, the system can acquire information of each individual user in real time. However, the quantity of the information becomes enormous. There occurs a problem that a human who control system cannot handle all the information. Our approach is that a human gives rough navigation to agents and the agents give precise navigation to each person. We aim at realizing a mega scale navigation system using GPS-capable cellular phones.

Figure 6 depicts the structure of the system. Important modules are described as below.

### 6.1. Control interface

The controller instructs the guiding agents the direction to evacuate through the control interface. In the interface, the map of a wide area is displayed so thatthe controller view the current locations of evacuees. The controller can also assign evacuation sites, set places ofshelters, and record the information about dangers such as fires.

On control interface, the distribution of people in the real space is reproduced on the virtual space with human figures based on positions of people acquired with sensors. The state of the virtual space is displayed on the monitor of the control center, so that the controller can grasp how people is moving in the real world widely through the bird-eye view of the virtual space. In addition, the controller can instruct particular people by handling human figures on the screen. The system notifies the people of the instructions using their registered phone numbers or e-mail addresses. Due to this interface, it is possible to grasp situations of all people with global view and provide local navigation with consideration of global coordination.

### 6.2. Guiding agents

Guiding agents guide evacuees in the disaster area. A guiding agent instructs the corresponding evacuee. These

[3] In this example, the ratio of the overhead to the execution time of cues and actions is estimated relatively large, because simple cues and actions are used. In real applications, cues and actions are more complex, and thus the ratio will be smaller.

**Figure 7. Screenshot of Large-Scale Evacuation Navigation System**

functions are implemented as functions of guiding agents. The behavior of the evacuee agent is given as the *Q* scenario.

Receiving location information from a user's GPS mobile phone, an agent sends a surrounding map according to the user's location. On this map, locations of places with damage such as fires, a location of a shelter to evacuate to, and a direction to head toward are described. The user sends his/her location and gets a new map when he/she needs.

An agent is instructed on a direction of evacuation by the control center. The agent retrieves shelters around the user, and selects a destination according to the ordered direction and distance between the user and each shelter. If the destination is changed by instructions, the agent notifies the user. If there is a person who needs rescue, his/her place is given to neighbor evacuees.

In this prototype, evacuee agents are given a simple uniform scenario. In future works, more complex navigation is provided by giving more variety of scenarios. Such scenarios will include ones that reflect social roles, such as firemen and police, individual contexts, such as injury, and so on.

## 7. Conclusion

In this paper, we have proposed an architecture for large-scale multiagent platform. We implemented a system that based on this architecture, evaluated it, and gave a sample application.

The problems we tackled in this work is as follows. The one is separation of protocol design and agent development. The architecture realizes the separation of protocol design and agent development, which enables the experts of different domains to cooperatively and efficiently develop large-scale multiagent system. The second is dynamic switching of protocols. By separating protocol processing system and agent internal models, experimenters can easily switch protocols according to the changing situations during operation. The third is scalability. By implementing both protocol processing system and agent internal models in a large-scale agent server, scalability of the system is improved.

The result of experiments shows that the Caribbean/*Q* system successfully manages 1,000,000 agents. However, to build more practical system, the speeding up is still necessary. To achieve it, technologies to distribute among multiple computers and to perform parallel is necessary. Besides the issue, we plan to study visualization methods of large-scale navigation system.

## Acknowledgment

We would like to thank Mr. Gaku Yamamoto and Mr. Hideki Tai at IBM Japan Tokyo Research Laboratory, and Mr. Akinari Yamamoto at Mathematical Systems Inc.. This work was supported by a JSPS Grant-in-Aid for Scientific Research (18200009), a Grant-in-Aid for JSPS Fellows and the Strategic Information and Communications R&D Promotion Programme.

## References

[1] L. Gasser and K. Kakugawa. Mace3j: fast flexible distributed simulation of large, large-grain multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 745–752, New York, NY, USA, 2002. ACM Press.

[2] O. Gutknecht and J. Ferber. The madkit agent platform architecture. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems: InternationalWorkshop on Infrastructure for Scalable Multi-Agent Systems, Lecture Notes in Computer Science*, volume 1887, pages 48–55. Springer, June 3-7 2001.

[3] T. Ishida. Q: A scenario description language for interactive agents. *Computer*, 35(11):42–47, 2002.

[4] T. Ishida. Society-centered design for socially embedded multiagent systems. In *International Workshop on Cooperative Information Agents (CIA-04), Lecture Notes in Computer Science*, volume 3191, pages 16–29, 2004.

[5] H. Kitano, S. Tadokor, H. Noda, I. Matsubara, T. Takahasi, A. Shinjou, and S. Shimada. Robocup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In *Proceedings of IEEE Conference on Systems, Men, and Cybernetics (SMC-99)*, volume VI, pages 739–743, Tokyo, Oct. 1999.

[6] Y. Murakami, Y. Sugimoto, and T. Ishida. Modeling human behavior for virtual training systems. *In the Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 127–132, 2005.

[7] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In *AOSE*, pages 121–140, 2000.

[8] G. Yamamoto and H. Tai. Performance evaluation of an agent server capable of hosting large numbers of agents. In *AGENTS-01*, pages 363–369, New York, NY, USA, 2001. ACM Press.