RoboCup Rescue Simulation Communication System version 1.231(RCRSCS)

1 Introduction

The RoboCup Rescue Agent Simulator is a discrete-time distributed simulation system and it aims to provide a simulation to reproduce conditions that arise after the occurrence of an earthquake in an urban area. In order to make the simulation more realistic, some issues are considered, such as heterogeneity (different types of rescue agents), dynamic environment (fires spread; injured victims), limited information (agents can only see a short distance), uncertain information (agents do not see the true state of the world), limited communication (messages can be dropped or have noise) and limited processing time (agents have a limited time to issue commands). It is thus characterized as a complex multiagent domain [Kitano et al., 1999].

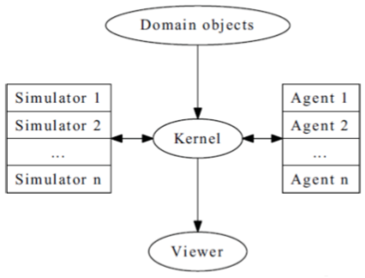

Fig. 1-1: RoboCup Rescue Agent Simulation platform architecture [Skinner and Ramchurn, 2010]

The architecture of the RoboCup Rescue Agent simulator (Fig. 1-1) is comprised of one kernel, one viewer, domain objects represented in the city map, some sub-simulators, each assigned to a specific task within the simulation, and some agents responsible to carry out tasks on the environment in order to mitigate the disaster impact [Skinner and Ramchurn, 2010]. There are sub-simulators responsible for fire simulation, traffic simulation, collapse simulation, blockage simulation and one responsible for miscellaneous simulations, such as humans ’ damage and buriedness and road clearing times. Each sub-simulator runs as an independent process, being the simulator kernel responsible for managing the interaction among them through the network. This setting allows for the computational load to be distributed among several computers, which helps the simulation workload distribution.

In order to perform their tasks, the agents are allowed to exchange messages among themselves through communication channels. The simulator allows the agents to use two types of messages: tell and say. In order to use the tell message type, the agent must be subscribed to a channel and when it sends a tell message through this channel, the content is broadcasted via radio to all the platoon agents subscribed to it. On the other hand, the say message type limits its receivers to all the agents located within a predefined radius from the sender.

Despite being able to communicate, the simulation platform does not define a protocol to be used for communication, letting the teams free to develop their own communication protocol in order to fulfill their strategic needs. However, such definition and development is time consuming, discouraging many teams to continue to develop their agent teams in order to participate in competitions. Furthermore, the communication protocol is not the teams ' goal when they first become interested by such simulation platform.

Therefore, the RoboCup Rescue Simulation Communication Library (RCRSCS) was developed by the SUNTORI team in order to minimize this problem allowing the teams to focus their efforts of the strategic part of their teams.

2 Concept

In disaster environments, there are different possible ways to control the individuals responsible to perform the actual tasks on the environment. One way of performing such coordination is by delegating it to a central coordinator who is responsible to make decisions about the tasks each individual should take. This coordination structure has several advantages and disadvantages. Among the advantages is the possibility for the central coordinator to select the best task available to each individual since it knows their positions and all the tasks available in the environment; however, this coordination strategy depends heavily on communication and create a single point-of-failure in the system.

Despite the drawbacks listed, it seems to be the most intuitive and simple form of coordination available. Therefore, the RCRSCS was implemented based on such coordination strategy in which the central location is intended to receive all the available information the individuals know from the environment and decide the task each individual should perform at each moment. Besides, the main aim of the library is to relief the teams from developing their own communication protocol before starting develop their team strategy.

Thus, technically speaking, platoon agents have a duty to perform actions based on the "Task Message" received from center agents. In addition, platoon agents send the obtained surrounding information (e.g. building, blockade, civilian, and so on) to other agents (platoon agents and center agents). The center agents have a duty to analyze the current situation from received information, and send "Task Message" to platoon agents as a result of their analysis. Moreover, the center agents consolidate the information in order to be able to make the bast possible decision about the task a platoon agent should perform [Suntori, 2012].

3 Install

You must download RCRSCS library from this URL, if you want to install this library,

- http://sourceforge.jp/projects/rcrscs/releases/

Once downloaded, you can extract it in any directory and add the rcrscs.jar extracted file into your build path in order to be able to use the RCRSCS communication library.

4 Usage

In order to use the RCRSCS communication library, you must:

1. Extend the AbstractCSAgent class instead of the StandardAgent when developing a platoon or center agent;
2. Override the following two methods
   a. protected EnumSet¡StandardEntityURN¿ getRequestedEntityURNEnum()
   b. protected void thinking(int time, ChangeSet change, Collection¡Command¿ heard)
3. Optionally, override the following method
   a. Void postConnect()

4.1 Example

```
public class simpleAmbulanceTeamAgent extends AbstractCSAgent<AmbulanceTeam>{
    @Override
    protected EnumSet<StandardEntityURN> getRequestedEntityURNsEnum(){
        return EnumSet.of(StandardEntityURN.AMBULANCE_TEAM);
    }
    /**
    * Implements the agent specific post connection code
    */
    @Override
    protected void postConnect(){
        ...
    }
    /**
    * Implements the agent specific processing
    */
    @Override
    protected void thinking(int time, ChangeSet change, Collection<Command> heard){
        ...
    }
}
```

There are two methods that should be used in order to enable the RCRSCS library functionalities. The methods are:

a. setMessageChannel(int channel)
   This method indicates to the RCRSCS library which channel the agent is going to use in order to send and receive messages. It should be used only once, since all the message sends and receives are going to use this channel as a reference.

b. addMessage(RCRSCSMessage msg)
   This method adds a message into a queue. At the end of the thinking method processing, the RCRSCS library automatically sends all the messages in this queue through the channel set using the setMessageChannel method. There are several types of RCRSCSMessage available, which is described in section Message.

The RCRSCS communication library makes available one attribute that allows the access to the heard messages from the channel set in the setMessageChannel. The attribute name is receiveMessageList and it is a list of RCRSCSMessage and it should be accessed directly in the agent code.

4.2 Internal Notes

The AbstractCSAgent extends the StandardAgent and implements the think method as required by the latter class.

```
@Override
/**
 * Represent each step thinking.<br>
 * (1.Receive message,2.think,3.send new messages)
 */
protected final void think(int time, ChangeSet changed, Collection<Command> heard) {
super.receiveMessage(heard);
this.thinking(time, changed, heard);
super.sendMessage(time);
}
```

Basically, what this code does is to filter all the messages in the heard collection and add to the receiveMessageList attribute the messages received from the channel set using the setMessageChannel method. Next, it executes the implemented thinking method from the agent. Finally, it sends all the messages added to the queue during the thinking processing to the channel set using the setMessageChannel.

5 Message

The RCRSCS communication library predefines three types of messages that can be sent: Information Messages, Task Messages and Report Messages.

5.1 Information Message

The Information Message represents information from the disaster space. It does not include static information from those objects in order to avoid the transmission of unnecessary data, thus reducing the size of message sent. They have specific meaning in the context of the RCRSCS communication library as described in the Table 5-1.

Table. 5-1: Information Messages description

Class	Description
FireBrigadeInformation	Fire Brigade information
PoliceForceInformation	Police Force information
AmbulanceTeamInformation	Ambulance Team information
BuildingInformation	Building information
BlockadeInformation	Blockade information
VictimInformation	Civilian information
TransferInformation	Pathway information
UnpassableInformation	It is impossible of traffic road information

The Information Messages and their elements are listed in the Table 5-2.

Table. 5-2: Information Message classes and parameters

FireBrigadeInformation(int time, EntityID fbId, int hp, int damage, int buriedness, int water, EntityID areaID)
PoliceForceInformation(int time, EntityID pfId, int hp, int damage, int buriedness, EntityID areaID)
AmbulanceTeamInformation(int time, EntityID atId, int hp, int damage, int briedness, EntityID areaID
BuildingInformation(int time, int EntityID buildingID, int fieryness, int brokenness)
BlockadeInformation(int time, int EntityID blockadeID, EntityID roadID, int repairCost)
VictimInformation(int time, EntityID vicID, EntityID area, int hp, int buriedness, int damage)
TransferInformation(int time, EntityID platoonID, EntityID... areas)
UnpassableInformation(int time, EntityID platoonID, EntityID from, EntityID to, EntityID blockade)

5.2 TaskMessage

The Task Message represents orders the platoon agents should perform. This message does not directly change the agent behavior, but it provides the direction the agents should follow to cooperate among the other agents. They have specific meaning in the context of the RCRSCS communication library as described in the Table 5-3.

Table. 5-3: Task Messages description

Class	Description
ClearRouteTaskMessage	Command a Police Force to clear roads from an area A to an area B
RescueAreaTaskMessage	Command a Ambulance Team to rescue a civilian
ExtinguishAreaTaskMessage	Command a Fire Brigade to extinguish fire from a specific building
ScoutAreaTaskMessage	Command a agent to scout to some area
DecideLeaderTaskMessage	Command a optional agent to provide center work when situation is centerless

The Task Messages and their elements are listed in the Table 5-4.

Table. 5-4: TaskMessage classes and parameters

ClearRouteTaskMessage(int time, EntityID ownerID, EntityID pfID, EntityID departure, EntityID destination)
RescueAreaTaskMessage(int time, EntityID ownerID, EntityID atID, EntityID... areas)
ExtinguishAreaTaskMessage(int time, EntityID ownerID, EntityID fbID, EntityID... areas)
ScoutAreaTaskMessage(int time, EntityID ownerID, EntityID targetID, EntityID... areas)
DecideLeaderTaskMessage(int time, EntityID ownerID, EntityID targetID)

5.3 ReportMessage

The Report Message reports the order results sent to the agents using a Task Message. The types of Report Message and their meanings are described in Table 5-5.

The Report Messages and their elements are listed in the Table 5-6.

Table. 5-5: Report Messages description

Class	Description
DoneReportMessage	This message indicates the agent has accomplished the task assigned to it
ExceptionReportMessage	This message indicates the agent was unable to complete the task assigned to it

Table. 5-6: Report Message classes and parameters

DoneReportMessage(int time, EntityID platoonID)
ExceptionReportMessage(int time, EntityID platoonID)

6 Refference

H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, volume 6, pages 739—743, Tokyo, Japan, October 1999. IEEE.
C. Skinner and S. Ramchurn. The robocup rescue simulation platform. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1, AAMAS ’10, pages 1647—1648, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
Suntori Team. Library for Communication — ReadMe. 2012.