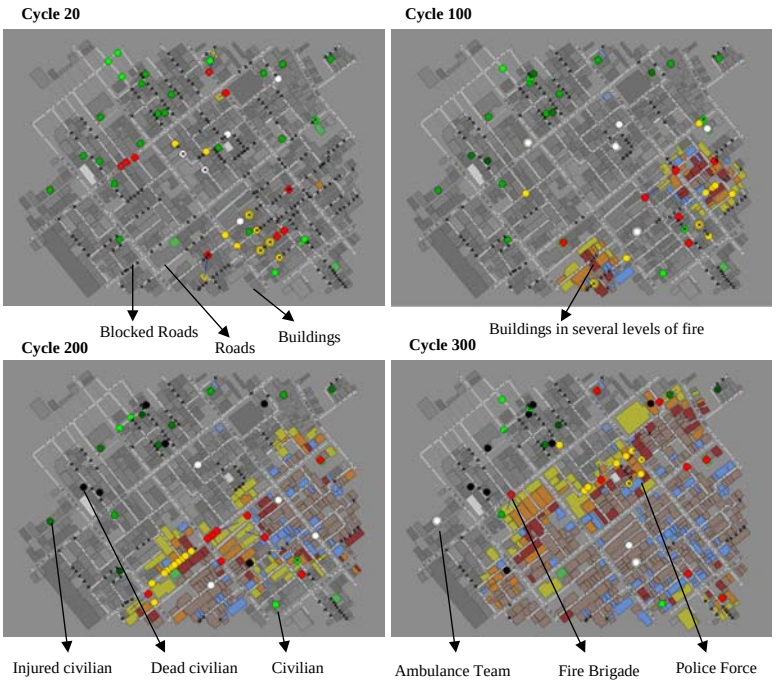
I-Kobe[1]

An I-X application to support disaster relief operations in the RoboCup Rescue Kobe scenario



Clauirton Siebra and Austin Tate
Artificial Intelligence Application Institute
Centre for Intelligent Systems and their Applications
School of Informatics, The University of Edinburgh
Appleton Tower, Crichton Street, EH8 9LE, Edinburgh, UK

I-X page: http://i-x.info
I-Rescue page: http://i-rescue.org

Date: 31-Aug-2005

[1] I-Kobe is part of the I-Rescue project, an effort towards the development of knowledge-based tools applied to search and rescue or disaster relief domains.

1. Kobe Domain: an example of urban disaster scenario

The occurrence of disasters in urban areas is the most critical event in terms of human lives due to the population concentration and considerable number of buildings. The *Great Hanshin Earthquake* or *Kobe Earthquake* is an example of how disasters have tragic effects in urban areas. On Tuesday, January 17th 1995, at 5.46 a.m. (local time), an earthquake of magnitude 7.2 on the *Richter Scale* struck the Kobe region of south-central Japan. This region is the second most populated and industrialised area after Tokyo, with a total population of about 10 million people. The ground shook for only about 20 seconds, but in that short time over 5,000 people died, over 300,000 people became homeless and damage worth an estimated £100 billion was caused to roads, houses, factories and infrastructure (gas, electric, water, sewerage, phone cables, etc). The use of knowledge-based tools in scenarios like that has the important goal of decreasing such tragic effects, extending the capabilities of human teams during search and rescue activities.

2. RoboCup Rescue Simulator

The *RoboCup Rescue* (RCR) simulator [1] is a real-time distributed simulation system that is built of several modules connected through a network via a central *kernel*, which manages communications among such modules. Each module can run on different computers as an independent program, so that the computational load of the simulation can be distributed across multiple processors. Each disaster phenomenon, such as collapse of buildings or fire spread, is simulated by a dedicated simulator-module, while a *Geographical Information System* (GIS) provides the initial condition of the disaster area.

The RCR simulator recognises six different types of agents: ambulance team, fire brigade, police force, ambulance centre, fire station and police office. Such agents act as several independent modules, which receive information about the environment and send commands to be performed by the kernel. In our implementation some of such agents are represented by I-X agents as detailed soon.

The simulation proceeds by repeating a specific cycle of one second, which corresponds to one minute in the real disaster space. However the rate can be configured according to the application that we intend to test. During each cycle the following steps are carried out:

- The kernel sends individual sensory information to each agent;
- Each agent individually submits an action command to the kernel;
- The kernel sends the action commands of agents to all sub-simulators;
- Sub-simulators submit updated states of the disaster space to the kernel;
- The kernel integrates the received states, sending the result to the viewer;
- The kernel advances the simulation clock of the disaster space.

The RCR disaster space is configured to represent the Kobe scenario. However we can model and use customised scenarios. For that end, there are appropriate graphical tools that generate geographical data in the format used by the RCR simulator. We have used one of these tools to create a scenario representing part of Nagoya city, for

example. Together with this capability of configuration, the principal advantage of using the RCR simulator is that it provides a dynamic and unpredicted environment, which enables the evaluation of different approaches in a more realistic way.

3. Application Architecture

The architecture that was used for this initial I-Kobe version is illustrated bellow (Figure 1). As introduced in the last section, the RoboCup Rescue simulator is the component that accounts for generating sensory information and processing the agents' commands. Sensory information from the simulator is indexed with the identification of agents and each of these agents only receives the perceptions associated with the scenario around it. Basically fixed agents (e.g., Police Office) do not receive new information about the environment because they do not change their positions. Action commands (e.g., rescue, extinguish, move, etc.) are only generated by the non-fixed agents: Ambulance Teams (AT), Fire Brigades (FB) and Police Forces (PF).

Agents are divided into two groups: RCR agents and I-X agents. RCR agents are provided in the simulator package (yapapi) and represent very simple agents that can be used as a basis for more complex implementations. I-X agents are the components that we have implemented and that are being evaluated inside the RCR environment.

The grey bars and arrows (Figure 1) represent the communication network, which defines which agents can interact. For example, fire brigades (FB_i) can only interact among themselves and with the fire station. For the experiments discussed here, the architecture uses a communication strategy in which each I-X agent is mapped to a host and port number, and messages are sent by writing their serialisations to a socket. A thread that acts as a name-server on a specified port accounts for the role of rotating messages between agents. Note, however, that the I-X architecture is independent of the communication strategy, so that other options could be used (e.g., *Jabber I-X Instant Messaging Technology*).

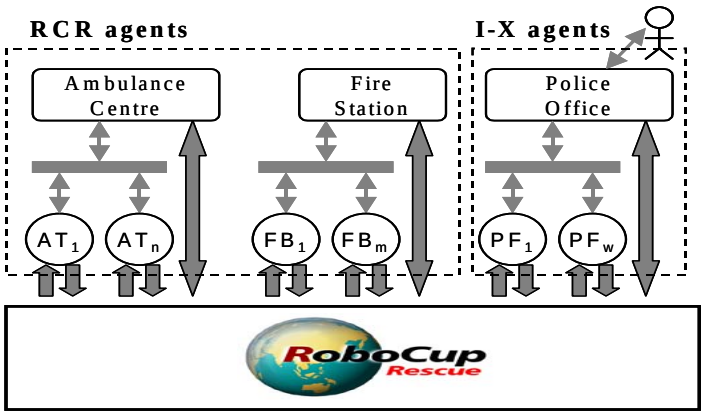


Figure 1. Application architecture

4. Quick Start

This section lists the steps that must be carried out to start the application. A first feature to be noticed is that the simulator only works on a Linux-based platform.

Differently, both RCR agents and I-X agents can run on any platform that supports the Java Runtime Environment. As any I-X application, all the necessary content of the I-Kobe can be found in the directory ix-<version>/apps. For the I-Kobe application, in particular, we have the following directory tree:

ix-<version>/apps/kobe/config
ix-<version>/apps/kobe/docs
ix-<version>/apps/kobe/domain-library
ix-<version>/apps/kobe/etc
ix-<version>/apps/kobe/imports
ix-<version>/apps/kobe/java
ix-<version>/apps/kobe/resources
ix-<version>/apps/kobe/scripts

We must set some parameters before starting the application. Such parameter are explained in Table 1.

Parameter	**Location**
Step	../kobe/etc/rescue<version>/boot/config.txt
Defines the duration of a cycle in milliseconds. Its value can be increased, for example, if you want to see the effect of the actions at each cycle, or have more time to make decisions. The current value is 5000	
host and port	../kobe/scripts/unix/2-start-rcr-agents.sh
Defines the host and port where the RCR agents must look for the simulator (kernel). The current values are 127.0.0.0 as host, and 8000 as port. This host is equivalent to the localhost and must be used if agents are going to run on the same host of the simulator. Port 8000 is the default value for UDP connections. TCP connections must use the port 7000.	
-rcr-hostname and –rcr-port	../kobe/scripts/unix/3-start-ix-agents.sh
- Similarly, defines the host and port where the I-X agents must look for the simulator (kernel).	
-action-message-time	../kobe/scripts/unix/3-start-ix-agents.sh
I-X agents that represent police forces can print a simple log to indicate the kind of actions (search, move and clear) they are taking during the simulation. This parameter set the period that such information is generated. The current value is 10.	

Table 1. Principal application parameters

The default number of agents of this application is:

1 Fire station (RCR agent);
10 Fire brigades (RCR agents)
1 Ambulance centre (RCR agent);
5 Ambulance teams (RCR agents);
1 Police office (I-P^2, I-X Process Panel [2]);
10 Police forces (I-X agents, no GUI);

However the number of agents can be changed. This process involves three files:

../kobe/etc/rescue<version>/boot/maps/Kobe/gisini.txt
../kobe/scripts/unix/2-start-rcr-agents.sh
../kobe/scripts/unix/3-start-ix-agents.sh

The gisini.txt file says to kernel the number of agents for the current simulation via the parameters: AmbulanceCenterNum, FireStationNum, PoliceForceNum, AmbulanceTeamNum, FireBrigateNum and PoliceForceNum. Note that if we define, for example, AmbulanceTeamNum=5, we must define 5 entries for each ambulance team in the same file. Examples of such entries can be found in the gisini.txt file.

In brief, the gisini.txt says how many agents are going to connect to the kernel, together with some initial information such as the initial position of each agent. After that, we must use the next two script files to start such agents. The important point here is that the number and kind of agents that will be started have to match the number defined in the gisini.txt.

Finally we can start the simulation by using the three scripts below:
../kobe/scripts/unix/1-simulator.sh
../kobe/scripts/unix/2-start-rcr-agents.sh
../kobe/scripts/unix/3-start-ix-agents.sh

Agents are connected in the simulator if they receive an *id* from the simulator. After the last agent receives this id, the simulation automatically starts.

5. The Simulation

The steps below give a general idea about what happens during the simulation progress:

1. Each police force sends its *id* and *current position* to the police office. This enables its representation by the I-X Map Tool;
2. Police forces decide by themselves what should be done if they do not receive any activity from the police office. Such behaviour is associated with the activities of search and clearing blocked roads;
3. The fire station accounts for receiving requests to clear roads from fire brigades and send these requests to the police office. Then, the police office agent transforms the requests in activities and update the map scenario so that it shows the blocked roads (road with a red cross);
4. The human user of the police office agent must decide which actions will handle the activities. Basically he/she can delegate activities to one of the police forces, or apply an allocation action (SimpleAllocatorHandler) to automatically choose the best agent(s) to perform the activities;
5. When an action is applied to the activity, such activity changes its status from *Ready* (orange colour) to *Executing* (green colour). After a police force executes the activity, it sends a completion report to the police office and the action's status is changed to *Complete* (blue colour);

Each police force agent generates a simple log saying which actions (search, clear or move) were performed during a specific period of time[2]. The *search* action means that the agent is looking for roads to be cleared. The *clear* action means that the agent is on the blocked road, clearing such blockage. The *move* action means that the agent is moving itself to a position indicated by the police office. We can say that the police

[2] Such period can be configured by the parameter *-action-message-time*, as explained in Table 1.

office's role is to optimise the performance of its police forces. This log can provide an idea about this performance because, in a general way, the goal is to increase the number of clear actions during a period of time, consequently decreasing the number of move and search actions.

6. The Simple Allocation Handler

The *SimpleAllocation* action is an example of an I-X handler that can be used by the police office. Its goals is, as cited before, to increase the number of clear actions during a period of time. For that, the handler is based on three ideas:

- Closest distance: the handler tries to allocate the closest agent to a blocked road. If this is not possible, it tries the second closest agent and so on;
- Loading balancing: the handler first tries the agents with the least number of activities;
- Multiple allocations: considering an activity *a*, the handler tries to allocate more than one agent to *a* (there is a constant X that limits the maximum number of agents that can be allocated to *a*) if there are agents with less executing activities than a constant Y. For this version we have X=2 and Y=1 as default values. Theoretically the code enables changes in such values, however this feature was not well tested yet.

The Java code that implements these ideas can be seen in:

ix-<version>/apps/kobe/java/ix/rcragents/handlers/SimpleAllocatorHandler.java

7. Limitations and Improvements

This first I-Kobe version implements the basic features about the integration between the RoboCup Rescue simulator and the I-X architecture. At the moment we are expanding this version so that it presents useful capacities associated with the development of human-agent collaborative planning. Some examples of limitations that can be found in this current implementation are:

- Police forces only report completion or failure of activities. Reports associated with activity commitments and progress are also important because they provide useful information to be used by the handlers, for example. Future versions will show how we can develop more powerful handlers if we have these reports defined by the architecture;

- In situations where the police office allocates a clear activity to two agents, two sub-nodes are created to represent such allocations. These nodes are typical examples of "or-activities" because only one of them needs to be completed to the clear activity be finished. However this doesn't happen in this version.

The theoretical proposal that we are implementing for future versions intends to provide a general (non-domain specific) solution for problems like that. In this way, the system will enable support to any other disaster relief domain or, in a broader perspective, any type of coalition domain.

Acknowledgement

Clauirton Siebra' scholarship is sponsored by CAPES Foundation under process number BEX2092/00-0. This material is based on research within the I-X project sponsored by the Defense Advanced Research Projects Agency (DARPA) and US Air Force Research Laboratory under agreement number F30602-03-2-0014 and other sources.

The University of Edinburgh and research sponsors are authorised to reproduce and distribute reprints and on-line copies for their purposes not withstanding any copyright annotation here on. The views and conclusions contained here in are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of other parties.

References

[1] Kitano, H. and Tadokoro, S. (2001). RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, 22(1):39-52.

[2] Tate, A., Dalton, J. and Stader, J. (2002) I-P^2 – Intelligent Process Panels to Support Coalition Operations. *Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations*, Toulouse, France, pp. 184-190.