RoboCup-Rescue Simulator Manual version 0 revision 4

The RoboCup Rescue Technical Committee

July 1, 2000

web page :
http://robomec.cs.kobe-u.ac.jp/robocup-rescue

mailing list :
rescue@symbio.jst.go.jp (japanese)
r-resc@isi.edu

Contents

1 Preface

The RoboCup-Rescue Project is starting for the research of disaster mitigation, search and rescue problems. Its objectives are

1. the robotics technology is applied to serious social problems to contribute human social welfare,
2. new practical problems are proposed as challenges of robotics and AI to initiate a novel research field, and
3. a new RoboCup competition is initiated in order to promote international research collaboration.

RoboCup-Rescue simulator aims to be a comprehensive urban disaster simulator, and is constructed on distributed computers.

- Heterogeneous intelligent agents such as fire fighters, victims and volunteers conduct search and rescue activities in this virtual disaster world.
- A real world interface integrates various sensor systems and controllers of infrastructures in the real cities with the virtual world. Real-time simulation is synchronized with actual disasters, computing complex relationship between various damage factors and agent behaviors.
- A mission-critical man-machine interface provides portability and robustness of disaster mitigation centers, and augmented-reality interfaces for rescue parties in real disasters.
- It will provide a virtual-reality training function for the public. This diverse spectrum of RoboCup-Rescue contributes to the creation of the safer social system.

On April 30 1999, a meeting was held at a corner of the hall where people were busy in preparing RoboCup Japan Open '99. At the meeting, it was decided to make a prototype of RoboCup-Rescue simulator by the end of 1999. Form that time, many people have been joined this project and a prototype will be releases this spring. We hope this manual will help more people join RoboCup-Rescue project.

2 Introduction

The Hanshin-Awaji Earthquake killed more than 6,500 citizens in Kobe on January 1, 1995. 80,000 wooden houses were fully collapsed and the number of sufferers were more than 1 million. The damage of the basic infrastructures exceeded 100 billion US dollars.

The experiences at the Hanshin-Awaji Earthquake concluded that the following functions were necessary to information systems for disasters.

1. Collection, accumulation, relay, selection, summarization and distribution of necessary information.
2. Prompt support for action planning of disaster mitigation, search and rescue.
3. Reliability and robustness of the system.
4. Continuity from ordinary times to emergency.

2.1 Nagata Ward Case for Prototype System

The RoboCup-Rescue project will support the functions for disaster and rescue simulations. Necessary conditions for rescue project are considered by investigating disasters in Nagata Ward, one of most damaged areas of the Hanshin-Awaji Earthquake.

1. **simulation period** Rescue activities start when earthquakes occur, and to be continued for years. The purposes of rescue activities change as reconstructions continue. The rescue activities are classified into five stages. Main purpose of rescue activities at the first stage is saving the victims. The period to be simulated is set the first **72** hours, considering that after that period survival rate decrease.
2. **rescue agents**: When earthquakes occurred, there are many calls for asking fire men. So rescue simulation will be done by local rescue agents. There were total **7** rescue agents – 5 firebrigates at the main fire office and 2 firebrigates at a branch fire office –.
3. **space resolution**: It is required to display things at the size of cars in order to represent disaster situations or rescue activities. GIS data is maintained with **5m** mesh.

   The area of 1.5 km^2 centered JR Nagata railway station is selected to be a target for the prototype system from the amount of GIS data.

Replacing Nagata ward's data with data for Los Angeles, Turkey, Taiwan, RoboCup Rescue simulator will simulate the disaster respectively.

Table 1: Statistical Data of Nagata Ward (at 1994 Oct.1)

area	11,47 km^2
household	53,284
population	130,466

2.2 formulation

2.2.1 one disaster simulation

Disaster situations changes every moment. The field called dynamic simulation handles methods how to simulate changing situations in computers.

Dynamic simulation is modeled by following equations.

$$\begin{aligned} \boldsymbol{e}(t) &= \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) && (1) \\ \boldsymbol{x}(t+\Delta t) &= \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{e}(t)) && (2) \end{aligned}$$

t is time, Δt is a step size to forward the simulation discretely. 1 second or 1 mill second are used in computation.

$\boldsymbol{x}(t)$ is called status variable. It represents disaster situation at time t. for example, the strength of fire, the speed of cars in traffic. Their values are different at places, so $\boldsymbol{x}(t)$ forms a vector. Its elements are values at different places, and the size becomes large as the simulated area become large.

$\boldsymbol{u}(t)$ is input at time t. The vector represents effects from the outside. They represent amount of water sprayed from fire engines, the direction and strength of wind, etc.

$\boldsymbol{f}$ is a function that describes how $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ at time t effect the simulated world, change its status. The effect calculated by function $\boldsymbol{f}$ is represented as $\boldsymbol{e}$.

$\boldsymbol{g}$ is also a function that describes the values of status $\boldsymbol{x}(t)$ at next time $(t+\Delta t)$.

2.3 disaster simulations

Each damage prediction (spread of fire, collapse of buildings, life line cut, etc.) needs large computational quantity on the basis of large scale data. Each damage has a relation to another. The activities of disaster mitigation, search and rescue also have a tight relation to the reduction of damage. Collapse of buildings affects spread of fire, fire fighters extinguish the fire, the fire spread forces to change the traffic condition, and traffic jams hamper the arrival of fire fighter parties.

Such the mutual relationship among disasters should be reflected synchronously to the conditions of simulation. The following equations represent the fundamental formation to calculate the effect of these disasters.

$$\begin{aligned}
\boldsymbol{e}_1(t) &= \boldsymbol{f}_1(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\
\boldsymbol{e}_2(t) &= \boldsymbol{f}_2(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\
&\vdots \\
\boldsymbol{e}_n(t) &= \boldsymbol{f}_n(\boldsymbol{x}(t), \boldsymbol{u}(t), t)
\end{aligned} \tag{3}$$

Each $\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_n$ represents one disaster simulation, such as fire simulation or traffic simulation. $\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n$ are associated effects. The next time situations are calculated by

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{g}(\boldsymbol{x}(t),\ \boldsymbol{e}_1(t), \boldsymbol{e}_2(t), \cdots, \boldsymbol{e}_n(t)). \tag{4}$$

2.4 simulation of agents behavior

The word *agent* is used in various contexts. In this manual, agents are used to refer autonomous entities in the simulated world, such as civilians, fire brigades, ambulance teams, etc. They decide their actions according to situations: go to refuges, ask for helps, extinguish fires, rescue victims, etc.

An agent's behavior is described in a form of algorithms with strategy. They are studied in fields like Artificial Intelligence, Robotics, Cognitive science, etc. The behavior of an agent is described as

$$\boldsymbol{e}_a(t) = \boldsymbol{f}_a(\boldsymbol{x}_a(t), \boldsymbol{s}(\boldsymbol{x}(t)),\ \boldsymbol{u}_a(t), t). \tag{5}$$

$\boldsymbol{x}_a$ represents the agent's ability such as knowledge, functions, the sense of value, etc. $\boldsymbol{e}_a(t)$ is the agent's work to the outside world. $\boldsymbol{s}$ is a function that restricts the input sensory data to data of circumstances around the agent. $\boldsymbol{u}_a(t)$ is the input to the agent from the outside world.

The agent's action changes outside world $\boldsymbol{x}(t + \Delta t)$.

$$\begin{aligned}
\boldsymbol{x}(t + \Delta t) &= \boldsymbol{g}(\boldsymbol{x}(t),\ \boldsymbol{e}_1(t), \boldsymbol{e}_2(t), \cdots, \boldsymbol{e}_n(t), \\
&\quad \boldsymbol{e}_{a1}(t), \cdots, \boldsymbol{e}_{aN}(t))
\end{aligned} \tag{6}$$

The agent's ability changes through communication with other agents, and by its learning.

$$\begin{aligned}
\boldsymbol{x}_a(t + \Delta t) &= \boldsymbol{g}_a(\boldsymbol{x}_a(t), \boldsymbol{s}(\boldsymbol{x}(t)),\ \boldsymbol{e}_1(t), \boldsymbol{e}_2(t), \cdots, \\
&\quad \boldsymbol{e}_n(t),\ \boldsymbol{e}_{a1}(t), \cdots, \boldsymbol{e}_{aN}(t))
\end{aligned} \tag{7}$$

These equations have the same structure as disaster simulations that are modeled as physical phenomena. From methodological view of a comprehensive disaster simulator, the simulations between disasters and agent behaviors are not different. It is possible to simulate autonomous agents which move in the simulated world in similar ways as to simulate disasters in the world.

3 RoboCup-Rescue **Simulation System**

3.1 Structure of the Simulation System

The RoboCup-Rescue simulation is built of a number of modules which communicate with each other using a protocol based upon UDP. The protocol is generic and does not depend on any particular simulation algorithm. The modules consist of **kernel**, **agents**, **component simulators**, **GIS** (Geographical Information System) and **viewers** (Fig. 1).

In the RoboCup-Rescue Simulation System, there exists only one kernel and one GIS, and there may be more than one agent, component simulator or viewer.

The terms – agent, simulator, modules – are used in the followings definition. And the term – module – is used to refer them.

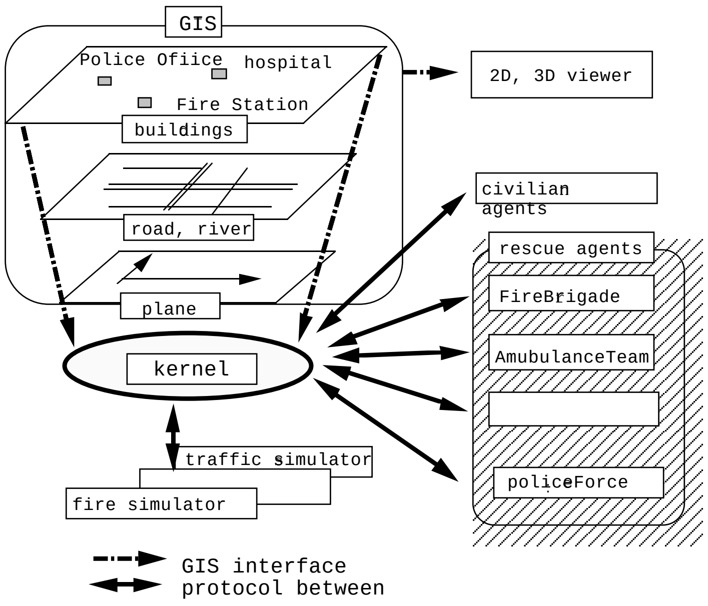

Figure 1: Overview of rescue simulator

Agent

The agent module controls an intelligent individual that decides its own action according to situations . Civilians, fire men, fire services and so on are agents [1] .

[1]Term – agent – is used to refer to agent module or to the individual, when there is no ambiguity.

Individuals are virtual entities in the simulated world. Their "will" and actions are controlled by the corresponding agents, which are the RoboCup-Rescue Simulation System's clients programs.

Kernel of simulator is a server program to the clients. The kernel module checks the actions. For example, a fire-fighter agent decides that it should extinguish a fire, but some trouble may prevent the fire plug from providing water.

Currently, an agent is a unit like a family or a fire brigade in order not to make the simulation system too large. In future, a single person (or a single robot) will be represented by an individual agent.

Component simulators

Component simulators correspond to various simulation domains, such as earthquakes, fires, logistics or traffic jams. The component simulators, which are plugged into the system, compute what will happen in the world, including what the individuals in the world will do and what the effects of their actions will be.

Only one component simulator can be plugged into the system for the same simulation domain. For example, one fire simulator simulates fire disasters [2] in the area. It should be noted that component simulators themselves may be participants of the simulation competition. The simulators compete with each other from the accuracy of simulation and its real time performance.

GIS

The GIS module provides the initial configuration of the world, where roads, buildings and individuals are located at the beginning of the simulation. It also records the simulation log, thereby offline scrutiny of the simulation progress will be possible.

Viewers

Viewers visualize the RoboCup-Rescue simulation via state-of-the-art computer graphics. They may also be evaluated at the competition.

Kernel

The kernel controls the simulation process and facilitates information sharing among the modules. In the future, the kernel should manage tens of thousands modules and their communication in real time. This will be one of the most challenging problems in the simulation system construction.

[2]Of course, there is another way to separate the simulation space by ares. In that case, different simulators simulate disasters at different areas.

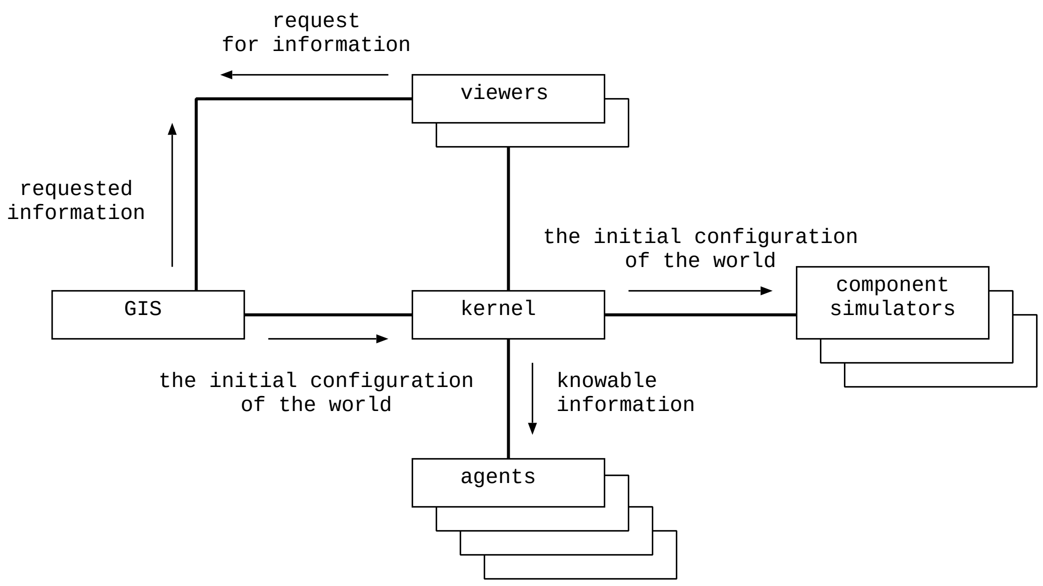

Figure 2: Communication at the beginning.

One module is represented by one process; however, it is possible to make a single process represent more than one module by using multiple threads or event-driven pseudo-processes. In particular, the inter-module communication protocol was designed so that more than one agent module can share a socket, since the number of agents will be large.

No two agents can communicate directly with each other. That is, agent modules can communicate with each other only by way of the kernel module, which accepts messages admissible in the communication protocol, as is the case in the RoboCup simulation. Other kind modules may communicate directly with each other; however, for the sake of modularity, plug-in component simulators are expected to communicate with others only by way of the kernel.

3.2 Progress of the Simulation

At the beginning of the simulation (Fig. 2), the GIS sends the kernel the initial configuration of the simulated world. The kernel then forwards this information to the component simulators, and sends to each agent modules all their knowable information.

In the prototype system one cycle taking one second of computer time simulates one minute of the real world time. The simulation cycle iterates the

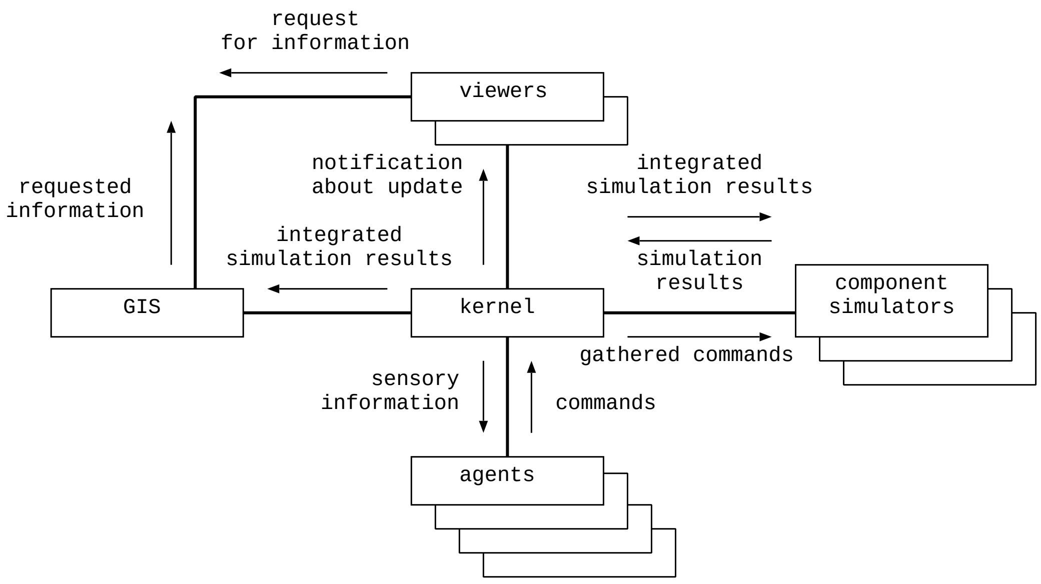

Figure 3: Communication among modules.

following steps (Fig. 3):

1. At the beginning of every cycle, the kernel sends **sensory information** to each agent module. This sensory information consists of information that the individual (controlled by the agent module) can sense in the simulated world at that time. The most of this information is visual information. The information may contain a certain amount of error as is the case in the RoboCup soccer simulation. In the prototype system, the kernel sends to agent modules data on all objects within a certain radius [3] .

2. Each agent module decides what actions the individual should take, and send it to the kernel. This message is called **command** (explained in section 3.4.1).

3. The kernel gathers all messages sent from agent modules, and broadcasts them to the component simulators. Commands are sometimes filtered. For example, commands sent by an agent module whose corresponding individual is already dead are discarded. Since the simulation proceeds in real time (sixty times faster than the real world clock, however), the kernel

[3]At present, the radius is 50m. The value will be set in config file.

ignores commands that do not arrive in time. Only accepted commands are broadcast to the component simulators.

4. The component simulators individually compute how the world will change based upon its internal status and the commands received from the kernel. These results are then sent back to the kernel.

5. The kernel integrates the results received from the component simulators, and broadcasts them to the GIS and the component simulators. The kernel will only integrate those results which are received within a certain deadline. The kernel then increases the simulation clock, and notifies the viewers about the update.

6. The viewers request the GIS to send updated information of the world, and display visually the information according to various evaluation criteria.

7. The GIS keeps track of the simulation results, and sends the viewers the information they request.

3.3 World Modeling

Component simulators may model the simulation world by their own methodologies. For example, some fire simulators may split the world into 50-meter planar grids, and some other fire simulators may have a 3D model of the town and houses. However, it is not easy for other modules to adapt to every variant or upgrade of the component simulator models. Therefore, the inter-module communication protocol is based upon a simple planar graph structure model with Euclidean metrics. For example, roads are described as edges of a graph with various useful information such as road width, and whether there is a median strip or sidewalks, associated with each edge. Each module has to communicate with others based upon this simplified world model, whether or not it uses a more sophisticated or finer-grained model.

The simple world model of the prototype system consists of various kinds of **objects** located on the graph, each of which has a certain number of **properties**. For example, an edge object that represents a road on the map has properties such as width and IDs of adjoint node objects, while a node object has its coordinates and IDs of edges it is connected to. Node objects represent crossroads, outstanding curves or linkage points between roads and building objects.

Modules do not send all the properties of each object every time; sending only necessary or limited part of properties of objects. For example, sensory information sent by the kernel to each agent contains only information which the individual can sense visually, aurally within its vicinity. Simulation results sent to the kernel from component simulators contain only the properties that will change at the next clock. However, all properties are only broadcast at the beginning of the simulation.

Objects in the prototype system other than edges and nodes are as follows:

Building : a building, which may collapse, obstruct a road, or bury persons alive. Its “entrance” property indicates a node in the graph where road is connected to the entrance of the building.

Civilian : a family which has encountered the disaster, and is being evacuated. It has properties such as coordinates, on what it is, HP (hit point), stamina, damage, etc.

Car : a car which is driven by a civilian. The properties are the same as civilian's properties.

Fire brigade : team that extinguishes fires. It has properties such as water quantity it can access, hose length it has, etc. other than those of civilians.

Fire station : station of fire brigades. It is a kind of building with other specific properties which the agent can control. It may control their fire brigades by means of wireless telephone, radio, or such.

Ambulance team : team that rescues persons buried alive and carries injured persons to hospitals. It will have properties representing its current capability.

Ambulance center : center of ambulance teams, similar to the fire station.

Police force : team that tries to open roads obstructed by debris, fissures or other kinds of obstacles.

Police office : center of police forces, similar to the fire station.

3.4 protocol

The protocol specifies communications among modules – kernel, agents, simulators, GIS and viewers –. One kernel component and one GIS component exist in the simulator, while components of other kind may exist. Agent components represent civilian [4] , fire fighters .etc. who move autonomously in the simulated are. GIS(Geographical Information system) provides all geographical data such as roads, bridges, houses, etc. of the world. Disasters occurred in the simulated world or their effect on the civiilan lifetraffics are simulated by simulator components. Viewer components display situation of the world And kernel component serves data sharing and controls the simulators.

The protocols is represented according to RFC1014 (XDR: External Data Representation Standard). `0x` is a prefix to represent hexadecimal notation.

[4]Ideally an agent is a human, however in prototype an agent is a group such as a family, a fire brigade.

Table 2: commands in agent's protocol.

Command	Information Transfer	Function
Initialization		
init	agent → kernel	Initialization of agents
Action Commands		
move	agent → kernel	Motion of agent body
act	agent → kernel	General term for disaster mitigation action, implemented ones are extinguish, rescue, load, unload, clear.
say	agent → kernel (agent)	Auditory information transmission
tell	agent → kernel (agent)	Via transmission line
Sensory information		
see	agent ← kernel	Visual information acquisition
hear	agent ← kernel (agent)	Auditory information acquisition
listen	agent ← kernel (agent)	Via transmission line

3.4.1 Commands

The agent modules make decision of their own behaviors according to diverse objectives. The objectives are mainly disaster mitigation. They search the victims, rescue them, evacuate them to safe places, provide some foods, etc. Their actions are transmitted to the distributed kernels to renew the world's conditions. Visual and auditory information is also exchanged via the kernel. A local language is defined for the agent-agent communication. An example of agent protocol is shown in Table 2 [5].

3.4.2 action command

- move *Route*

  Move along *Route*. *Route* is an ID list consisting of connected roads and nodes.

- say *Target Message*

  Say *Message* to an individual whose ID is *Target*. *Message* is any string that is allowed by the protocol specification.

[5]Commands in this table were designed at June 1999. All of the commands were not implemented in version 0. The commands that have been implemented really are explained in section 9. Please refer to the section at programming.

- tell *Message*

  Broadcast *Message* by radio or other means.

- extinguish *Target*

  Extinguish a fire. *Target* is the ID of a burning building, or else.

- rescue *Target*

  Rescue an individual buried alive. *Target* is the ID of the individual.

- load *Target*

  Take on an individual. *Target* is the ID of the individual. Only one individual can be accommodated in the ambulance.

- unload

  Drop the individual.

- clear *Target*

  Restore a blockaded road so that cars can pass. *Target* is the ID of the road.

3.4.3 sensory information

The agent modules in the prototype system receive the following sensory information:

- sense *Receiver Time Self Objects*

  Inform the agent modules of visual and auditory information sensed by the corresponding individual. *Receiver* is the ID of the individual controlled by the agent module. *Time* indicates the current time. *Self* is the information about the individual. *Objects* is the information about the objects sensed at the time. It is a list of the information whose format is the same as that of *Self.*

  The format of *Self* is: *ID Property Property ...* *ID* is an ID number of the object. The format of *Property* is: *PropertyID Value. PropertyID* is the number which denotes the kind of the property. *Value* is the value of the property.

- hear *Receiver Time Sender Message*

  Inform the agent module of a message said or told by another individual. *Time* indicates the current time. *Sender* is the ID of the individual that said or told *Message.*

3.5 miscsimulator

At March 2000 when Rescue simulator was tested, various kinds of themess were discussed . (see.section 11) The role of kernel is one of the themes. In discussing it, it is concluded that the management of modules and the management of agents' status are different. A new simulator – miscellaneous simulator (miscsimulator) – is introduced to play the role of civilan agents status. The misc-simulator has been implemented from kernel release verion 0.23.

For example, when the civilian is in the collapsed house or it suffers from fire, the misc-simulator sets properties fo civilians. The present algorithm is a liner model: $hp = damage \times T + 10000$, where damage coefficcinet are -100 (in house collapsed) and -1000(fire), T is elapsed time. When the civilian pass the **refuge**, its damage is set 0. And, a firebrigade is an agent derived class of civilian. A firebrigade also suffers from damages at paresent.

The followings are items of miscsimulator handles

- commands: (see 3.4.2)

  load

  unload

  rescue buriedness of *Target* is decreased by 1.

  stretch A firebrigade pulls a fire hose to a fireplug. At present, fireplugs are not specified, so this one is reserved for the future.

  move

  clear

- property (see 4.2)

  hp Initilal value is 10000. The value is decreased by damage every step. Value '0' means dead.

  damage The value is set when the civilian suffers from fire or building collapses. (Tentatively, 40 is set in a case of fire and 400 is set in a case of building collapses, at June 2000.) When the civilian goes to a refuge, the value is set '0'.

  buriedness

4 Objects in the simulated world

This chapter describes objects in the simulated world. Kernel models the simulated world as a collection of objects. The objects are families, fire fighters and also houses and roads. The houses and roads are represented as a graph, and house objects and road objects are nodes and edges of the graph. Other components [6] should use the model, the collection of objects, when they communicate data with kernel component.

At Kernel version.0, the following objects [7] are implemented. A unique **ID** (a positive integer) is assigned to each object

4.1 World

The real longitude and latitude are used to locate positions of objects in GIS. The position data of simulated are translated into a new coordinate the origin of which is specified to a point in the simulated area.

property	value	comment
startTime	integer: −0x7FFFFFFF ∼ 0x7FFFFFFF	Simulation start time. (The time is elapsed minutes from Jan. 1. 1970 0:0 AM.)
longitude	integer: −0x7FFFFFFF ∼ 0x7FFFFFFF	The longitude of the new coordinates origin. The east longitude is positive, west longitude is negative. The unit is second degree.
latitude	integer: −0x7FFFFFFF ∼ 0x7FFFFFFF	The latitude of the new coordinates origin. The north latitude is positive, south longitude is negative. The unit is second degree.
windForce	integer: 0 ∼ 0x7FFFFFFF	current force of the wind. unit 0.001 meter/hour.
windDirection	integer: 0∼ 1295999	current direction of the wind. The positive direction along y axis is 0, the value moves to 1295999=360*60*60-1 seconds.

The properties – hp (hit point) and damage – are set by **misc-simulator** (see. setion 3.5).

[6] As internal models, the components may use any kind of models.

[7] Objects and properties marked by * are preserved for future use, and not used in the protocol explained later.

4.2 Civilian (agent)

property	value	comment
position	object ID or 0	ID of an object that the civilian is on or in. When the civilian is on a road, ID is the road's ID. It is in a building, ID is the building's ID. It is in an ambulance, ID is the ambulance's ID, etc. ID=0 means it is on/in nothing. Notice: Following the *position* properties must not reach itself. (A directed graph made from *position* properties closed paths.)
positionExtra	integer: 0 ∼ 0x7FFFFFFF	When the civilian is on the road, this property signifies the position on the road. The value is the distance form the head of the road. The range of the values is from 0 to the length of the road. The civilian is on/in other object, the value is set 0.
stamina	integer: 0 ∼ 0x7FFFFFFF	At this version, stamina remains constant. In future version, the agent's stamina will be decreased by taking actions, and the agent cannot take an action that causes the value of stamina minus. Kernel will restore a specified value every cycle.
hp	integer: 0 ∼ 0x7FFFFFFF	Kernel decreases this property's value by *damage* every cycle. 0 value is dead.
damage	integer: 0 ∼ 0x7FFFFFFF	This property shows the necessity of medical treatment. The value is decreased by treatment.
buriedness	integer: 0 ∼ 0x7FFFFFFF	This property shows how much the civilian is buried in the collapse of buildings. The value is how many people are required to save it form the collapse. The value more than one means he cannot move by himself. Default initial value is 0, and earthquake simulator updates the value.
direction	integer: 0 ∼ 1295999	The positive direction along y axis is 0, and the value moves to 129599=360*60*60-1 seconds counter clockwise.
positionHistory	a list of IDs	A list of IDs, such as houses or roads, that it passed during the previous one cycle. The order is chronological.

4.3 Car (agent)

Car agents are cars driven by civilian agents. Their properties are the same one as civilian's.

4.4 FireBrigade (agent)

FireBrigade agents have the following properties in additionto civilian agent's property.

property	value	comment
waterQuantity	integer: 0 ~ 0x7FFFFFFF	This property shows how much water is in the tank.
stretchedLength	integer: 0 ~ 0x7FFFFFFF	This property shows how long the hose is pulled to the nearest.

4.5 AmbulanceTeam (agent)

AmbulanceTeam agents have the same properties as civilian agent.

4.6 PoliceForce (agent)

PoliceForce agents have the same properties as civilian agent. They restore collapsed roads to states that cars or civilian can pass.

4.7 Road (edge of road network)

Roads are represented with a graph which edges are roads and nodes are crossings.

property	value	comment
head	node ID or Building ID	ID of an end point of the road.
tail	node ID or Building ID	ID of the other end point.
length	integer: 0 ~ 0x7FFFFFFF	The length of the road, the unit is mm.
roadKind	integer: 0 ~ 0x7FFFFFFF	The vales, (0x01, 0x02, 0x04, 0x08), indicate the kind of road, (elevated road, bridge, tunnel, road for emergency), respectively.
carsPassToHead	integer: 0 ~ 0x7FFFFFFF	The number of cars that passed during the previous one cycle from tail to head.
carsPassToTail	integer: 0 ~ 0x7FFFFFFF	The number of cars that passed during the previous one cycle from head to tail.
humansPassToHead	integer: 0 ~ 0x7FFFFFFF	The number of human cars that passed during the previous one cycle from tail to head.
humansPassToTail	integer: 0 ~ 0x7FFFFFFF	The number of human that passed during the previous one cycle from head to tail.
width	integer: 0 ~ 0x7FFFFFFF	The width of the road, the unit is mm.
block	integer: 0 ~ 0x7FFFFFFF	The width of a part of the road where cars or human cannot pass by collapse of buildings, cracks etc.,, the unit is mm.
repairCost	integer: 0 ~ 0x7FFFFFFF	This property shows how many people are required to restore the road to the normal that *block* is 0.
medianStrip	0 or 1	The value is 1 when there is a median strip, otherwise 0.
linesToHead	integer: 0 ~ 0x7FFFFFFF	The number of traffic lanes form tail to head.
linesToTail	integer: 0 ~ 0x7FFFFFFF	The number of traffic lanes form head to tail.
widthForWalkers	integer: 0 ~ 0x7FFFFFFF	The width of a part of the road for pedestrians.

4.8 Node (vertex of road network)

property	value	comment
x	integer: 0 ~ 0x7FFFFFFF	x coordinate
y	integer: 0 ~ 0x7FFFFFFF	y coordinate
edges	IDs of roads or buildings	a set of ID of objects, roads or building, that connect to this node.
signal	0 or 1	The value is 1 when there are signals, otherwise 0.
shortcuToTurn	list of intergers: 0 ~ 0x7FFFFFFF	In a case of left (right) traffic system: A list of the number of short cut for left (right) turn. The numbers are in the order of the *edge*.
pocketToTurnAcross	list of integers: 0 ~ 0x7FFFFFFF	Under left (right) traffic system: A list of the numbers of pockets and length for right (left) turn. The numbers are in the order of the *edge*.
signalTiming	list of integers: 0 ~ 0x7FFFFFFF	A list of the time periods of signal for every road in the *edge*. The number is triplet for blue, yellow, and blue for right turn. The numbers are in the order of the *edge*.

4.9 River (edge of river network)

property	value	comment
head	node ID of river network	ID of an end point of river edge.
tail	node ID of river network	ID of the other end point.
length	integer: 0 ~ 0x7FFFFFFF	length of riveredge. unit is mm.

4.10 RiverNode (vertex of river network)

property	value	comment
x	0 ~ 0x7FFFFFFF integer:	x coordinate
y	0 ~ 0x7FFFFFFF integer:	y coordinate
edges	River IDs	a set of IDs that connects this RiverNode.

4.11 Building (buildings or building sites)

property	value	comment
x	integer: 0 ~ 0x7FFFFFFF	x coordinate, unit: mm (the same as the unit of *road*).
y	integer: 0 ~ 0x7FFFFFFF	y coordinate, unit: mm (the same as the unit of *road*).
floors	integer: 0 ~ 0x7FFFFFFF	the number of floors
buildingAttributes	integer: 0 ~ 0x7FFFFFFF	values - 0, 1, 2 - represent the kind of building, - a wooden house, a steel frame house, a reinforced concrete house - respectively.
ignition	0 or 1	The value is 1 for the building that Fire simulator sets fire to, otherwise 0.
fieryness	integer: 0 ~ 0x7FFFFFFF	How much it burns
brokenness	integer: 0 ~ 0x7FFFFFFF	The value indicates how much it collapsed. 0: no damaged, 25: partly damaged, 50: half collapsed, 100: fully collapsed.
entrances	ID of road network's Node	A list of ID of objects that the entrances of the building connect to.
buildingShapeID	integer: 0 ~ 0x7FFFFFFF	figure ID of building
buildingCode	integer: 0 ~ 0x7FFFFFFF	structure code
buildingAreaGround	integer: 0 ~ 0x7FFFFFFF	area of the first floor
buildingAreatotatal	integer: 0 ~ 0x7FFFFFFF	total floor area
buildingApexes	list of coordinates	coordinate of polygon's vertexs

4.12 Refuge

A king of buiding that civilians take refuge in. It has the same properties of building object.

4.13 FireStation

A buiding that can act autonoumously, namely, people in the building send command to FireBrigate. It has the same properties of building object.

4.14 AmbulanceCenter

A king of buiding that can act autonoumously, namely, people in the building send command to AmbulanceTeam. It has the same properties of building object.

4.15 PoliceOffice

A king of buiding that can act autonoumously namely, people in the building send command to PoliceOfiice. It has the same properties of building object.

4.16 sending or receiving format of Object information

The following formats are used at sending or receiving information of objects.

```
/*XDR*/
enum Property {
    PROPERTY_NULL = 0,

    PROPERTY_START_TIME = 29,
    PROPERTY_LONGITUDE = 30,
    PROPERTY_LATITUDE = 31,
    PROPERTY_WIND_FORCE = 32,
    PROPERTY_WIND_DIRECTION = 33,

    PROPERTY_X = 3,
    PROPERTY_Y = 4,

    PROPERTY_DIRECTION = 27,
    PROPERTY_POSITION = 6,
    PROPERTY_POSITION_HISTORY = 207,
    PROPERTY_POSITION_EXTRA = 7,

    PROPERTY_STAMINA = 9,
    PROPERTY_HP = 10,
    PROPERTY_DAMAGE = 11,
    PROPERTY_BURIEDNESS = 23,
    PROPERTY_FLOORS = 14,
    PROPERTY_BUILDING_ATTRIBUTES = 15,
    PROPERTY_IGNITION = 48,
    PROPERTY_BROKENNESS = 17,
    PROPERTY_FIERYNESS = 16,
    PROPERTY_ENTRANCES = 21,
    PROPERTY_BUILDING_SHAPE_ID = 49,
    PROPERTY_BUILDING_CODE = 50,
    PROPERTY_BUILDING_AREA_GROUND = 51,
    PROPERTY_BUILDING_AREA_TOTAL = 52,
    PROPERTY_BUILDING_APEXES = 53,

    PROPERTY_WATER_QUANTITY = 25,
    PROPERTY_STRETCHED_LENGTH = 26,

    PROPERTY_HEAD = 12,
    PROPERTY_TAIL = 13,
    PROPERTY_LENGTH = 24,

    PROPERTY_ROAD_KIND = 19,
    PROPERTY_CARS_PASS_TO_HEAD = 34,
    PROPERTY_CARS_PASS_TO_TAIL = 35,
    PROPERTY_HUMANS_PASS_TO_HEAD = 36,
    PROPERTY_HUMANS_PASS_TO_TAIL = 37,
    PROPERTY_WIDTH = 38,
    PROPERTY_BLOCK = 22,
    PROPERTY_REPAIR_COST = 39,
    PROPERTY_MEDIAN_STRIP = 40,
    PROPERTY_LINES_TO_HEAD = 41,
    PROPERTY_LINES_TO_TAIL = 42,
    PROPERTY_WIDTH_FOR_WALKERS = 43,

    PROPERTY_EDGES = 242,

    PROPERTY_SIGNAL = 44,
    PROPERTY_SIGNAL_TIMING = 194,
    PROPERTY_SHORTCUT_TO_TURN = 192,
    PROPERTY_POCKET_TO_TURN_ACROSS = 193,
};

/* a set of ID */
union IDs switch(int id) {
case 0:
    void;
default:
    IDs next;
```

```
};

/* a list of (property's kind , its value) */
union Properties switch(Property property) {
case PROPERTY_NULL:
    void;
default:
    int value;
    Properties next;
case /* value in a range {0x80, 0xBF} */:
    opaque value<>;
    Properties next;
case /* value in a range {0xC0, 0xFF} */:
    IDs value;
    Properties next;
};

/* Object's kind */
enum Type {
    TYPE_NULL = 0,

    TYPE_CIVILIAN = 232,
    TYPE_FIRE_BRIGADE = 233,
    TYPE_AMBULANCE_TEAM = 234,
    TYPE_POLICE_FORCE = 235,

    TYPE_ROAD = 168,
    TYPE_NODE = 200,

    TYPE_RIVER = 169,
    TYPE_RIVER_NODE = 201,

    TYPE_BUILDING = 176,
    TYPE_REFUGE = 184,
    TYPE_FIRE_STATION = 185,
    TYPE_AMBULANCE_CENTER = 186,
    TYPE_POLICE_OFFICE = 187
};

/* an Object */
struct Object {
    Type type;
    int id;    /* ID of this object */
    Properties properties;
};

/* Object more than zero */
```

```
union Objects switch(Type type) {
case TYPE_NULL:
    void;
defualt:
    int id;    /* ID of this object */
    Properties properties;
    Objects next;
};
```

It is possible to send the object's part by sending only properties that changed at the previous cycle. The data is said to be **complete** when it contains all properties. Future version ups may change the properties of objects, so the followings are recommended when receiving complete data.

1. When some of received properties are unknown, they are to be neglected.
2. Receiving an object at first time, their values should be set initial values.

There are some notes on receiving data:

1. ID of an object is uniquely set by *kernel*, however, it does not mean that IDs are the same to other *agent* modules. If ID is the same to all *agent* modules, victim agents can be easily identified among *agent* modules.
2. The values of properties may not be correct when *agent*s receive them. Because it may not be true that the data is always correct in disastered cites. (The values are correct at version 0.)

5 LongUDP

Communication between *kernel* and other *components* are done mostly by **UDP**. The data from *GIS* is a big one, and its length may be larger than 64kbyte that UDP can not handle. RoboCup-Rescue protocol provides **LongUDP** protocol to transmit a big packet. LongUDP divides the big packet into small parts, adds 8byte-header to each part, and send them by UDP. The header is the following format. The data in the head is represented in network byte order.

OFFSET	DATA	COMMENT
0	0x0008	magic number.
2	ID	ID number of LongUDP packet.
4	number	this integer shows where this UDP packet is in LongUDP. (0 ~ total−1).
6	total	total packet numbers in LongUDP packet.

LongUDP uses IP address and port number as well as UDP. The IP address and port number are the same ones as UDP. ID number in the LongUDP is assigned not to be the same as other LongUDP's ID. The long packet is unified by

1. collecting LongUDP packets with the same *ID* number,
2. after receiving *total* packets, sorting them in *number* ascending order,
3. concatenating them without header part.

The length of divided small parts except the last one (its *number*=*total*-1) is a multiple of four. The length of divided small parts is bigger than eight, i.e., they have data besides header part.

When *total* packets are not read for some period, some UDP packets may be lost. It is recommended to stop receiving the LongUDP packet. Otherwise a wrong Long UDP may be constructed later, because *kernel* creates ID number cyclically.

5.1 LongUDP packet format

A LongUDP packet has block more than 0. The block is composed of *header* and *body.*

```
/*XDR*/
enum Header {
    HEADER_NULL = 0,
    AK_CONNECT = 0x10,
    AK_ACKNOWLEDGE = 0x11,
    AK_REST = 0x80,
    AK_MOVE = 0x81,
    KA_CONNECT_OK = 0x50,
    KA_CONNECT_ERROR = 0x51,
    KA_SENSE = 0x52,
    SK_CONNECT = 0x20,
    SK_ACKNOWLEDGE = 0x21,
    SK_UPDATE = 0x22,
    KS_CONNECT_OK = 0x60,
    KS_CONNECT_ERROR = 0x61,
    KS_COMMANDS = 0x62,
    KS_UPDATE = 0x63,
    VK_CONNECT = 0x20,
    VK_ACKNOWLEDGE = 0x21,
    KV_CONNECT_OK = 0x60,
    KV_CONNECT_ERROR = 0x61,
    KV_UPDATE = 0x63,
    GK_CONNECT_OK = 0x??,
    GK_CONNECT_ERROR = 0x??,
    KG_CONNECT = 0x??,
    KG_ACKNOWLEDGE = 0x??,
    KG_UPDATE = 0x63,
};

union LongUDPPacket switch(int header) {
case HEADER_NULL:
    void;
default:
    opaque body<4294967294>;  /* body<0xFFFFFFFE> */
    LongUDPPacket next;
}
```

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 81	header=0x81
4	00 00 00 10	length of body=16
8	00 00 02 56	body
12	00 00 01 34	
16	00 00 01 6F	
20	00 00 00 00	
24	00 00 00 81	header=0x81
28	00 00 00 10	length of body=16
32	00 00 02 57	body
36	00 00 01 55	
40	00 00 01 A8	
44	00 00 00 00	
48	00 00 00 00	header=HEADER_NULL

The length of body may be 0xFFFFFFFF in the following example when kernel received the message. In this case, the length of body is calculated from the remaining packet's length.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 81	header=0x81
4	00 00 00 10	length of body=16
8	00 00 02 56	body
12	00 00 01 34	
16	00 00 01 6F	
20	00 00 00 00	
24	00 00 00 81	header=0x81
28	FF FF FF FF	length of body=16
32	00 00 02 57	body
36	00 00 01 55	
40	00 00 01 A8	
44	00 00 00 00	
48	00 00 00 00	header=HEADER_NULL

The body's content varies as its header. Connecting modules with different version may add extra data the body as explained later. The modules are recommended to neglect them.

Or broadcasting data cuases receiving packets for other module will cause similar situations. It also is recommended to neglect the packet for which the value of the header may be out of the specified values.

6 connection *kernel*

As explained in section 3.2, *kernel* plays an important role in exchanging data. Starting simulation, *kernel* connects *GIS*, *simulators*, *viewers*, and *agents* in order. Before connecting *agent*s, other modules are connected to *kernel. Kernel* starts simulation after all objects are connected.

During simulation, *kernel* repeats the following steps.

1. (from second cycle) send sensory information to agents,
2. (from second cycle) receive commands from agents,
3. forward them to simulators,
4. receive updated data from simulators,
5. update properties of agents,
6. send updated properties to simulators,
7. send the updated data to reviewers,
8. send the updated data to reviewers,
9. advance time one cycle.

At 2 and 4 step, *kernel* proceeds the next step in 0.5 seconds. Otherwise, finishing the step proceeds the next step. In the following section, the steps are explained more detailed.

6.1 connection to GIS

Kernel requires connection with *GIS* via port 6001.

```
header = KG_CONNECT
```

Its body has the following format.

```
/*XDR*/
struct KGConnect {
    int version;    /* must be 0 */
    /* if there is extra data added, they should be neglected. */
};
```

A UDP packet sent from *kernel* for connection has the following format.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 ??	header=0x??
4	00 00 00 04	length of body=4
8	00 00 00 00	version=0
12	00 00 00 00	header=HEADER_NULL

GIS returns GK_CONNECT_OK when the connection succedes, returns GK_CONNECT_ERROR in a case of failure. *Kernel* returns KG_ACKNOWLEDGE when it receives GK_CONNECT_OK.

GIS sends GK_CONNECT_OK again, when it does not receive KG_ACKNOWLEDGE after a specified time [8].

GK_CONNECT_OK's header and body are the followings.

```
header = GK_CONNECT_OK


/*XDR*/
struct GKConnectOk {
    Objects map;
};
```

The object *map* contains information of all objects in simulated world, and represents the initial status of simulation.

GK_CONNECT_ERROR's header and body are the followings. The string *reason* shows why it failed.

```
header = GK_CONNECT_ERROR


/*XDR*/
struct GKConnectError {
    string reason<255>;
};
```

KG_ACKNOWLEDGE's header and body are the followings.

```
header = KG_ACKNOWLEDGE


/*XDR*/
struct KGAcknowledge {
    void;
};
```

The following table shows addresses for sending and receiving.

header	send form	send to
KG_CONNECT	any	GIS (port 6001)
GK_CONNECT_OK	any	sender of KG_CONNECT
GK_CONNECT_ERROR	any	sender of KG_CONNECT
KG_ACKNOWLEDGE	any	sender of GK_CONNECT_OK

[8]Now the time is set 1 minute. At future version, the value is set in the config file. This situation is true for KS_CONNECT_OK, KV_CONNECT_OK, KA_CONNECT_OK cases.

6.2 connection to *Simulator*

Simulators require connection with *kernel* via port 6000.

```
header=SK_CONNECT
```

Its body has the following format.

```
/*XDR*/
struct SKConnect {
    int version;    /* must be 0 */
};
```

A UDP packet for connection has the following fromat.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 20	header=0x20
4	00 00 00 04	length of body=4
8	00 00 00 00	version=0
12	00 00 00 00	header=HEADER_NULL

Kernel returns KS_CONNECT_OK when the connection succeeds, otherwise returns KS_CONNECT_ERROR. *Simulators* returns SK_ACKNOWLEDGE, it receives KS_CONNECT_OK.

Kernel sends KS_CONNECT_OK again, when it does not receive SK_ACKNOWLEDGE after a specified time.

KS_CONNECT_OK's header and body are the followings.

```
header = KS_CONNECT_OK


/*XDR*/
struct KSConnectOk {
    Objects map;
    /* if there is extra data added, they should be neglected. */
};
```

The object *map* provides all information of simulated world that *kernel* has.

KS_CONNECT_ERROR's header and body are the followings. The string *reason* shows why it fails.

```
header = KS_CONNECT_ERROR


/*XDR*/
struct KSConnectError {
    string reason<255>;
    /* if there is extra data added, they should be neglected. */
};
```

SK_ACKNOWLEDGE's header and body are the followings.

```
header = SK_ACKNOWLEDGE
```

```
/*XDR*/
struct SKAcknowledge {
    void;
};
```

The following table shows addresses for sending and receiving.

header	send from	send to
SK_CONNECT	any	kernel (port 6000)
KS_CONNECT_OK	any	sender of SK_CONNECT
KS_CONNECT_ERROR	any	sender of SK_CONNECT
SK_ACKNOWLEDGE	sender of SK_CONNECT	sender of KS_CONNECT_OK

6.3 connection to Viewer

Viewer requires connection with *kernel* via port 6000.

```
header=VK_CONNECT
```

Its body has the following format.

```
/*XDR*/
struct VKConnect {
    int version;    /* must be 0 */
};
```

A UDP packet for connection has the following fromat.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 20	header=0x20
4	00 00 00 04	length of body=4
8	00 00 00 00	version=0
12	00 00 00 00	header=HEADER_NULL

Kernel returns KV_CONNECT_OK when the connection succedes, otherwise returns KV_CONNECT_ERROR. *Viewer* returns SK_ACKNOWLEDGE, when it receives KS_CONNECT_OK.

Kernel sends KV_CONNECT_OK again, when it does not receive VK_ACKNOWLEDGE after a specified time.

KV_CONNECT_OK's header and body are the followings.

```
header = KV_CONNECT_OK


/*XDR*/
struct KVConnectOk {
    Objects map;
    /* if there is extra data added, they should be neglected. */
};
```

The object *map* provides all information of simulated world that *kernel* has.

KV_CONNECT_ERROR's header and body are the followings.

```
header = KV_CONNECT_ERROR


/*XDR*/
struct KVConnectError {
    string reason<255>;
    /*  if there is extra data added, they should be neglected. */
};
```

The string *reason* shows why it fails.

VK_ACKNOWLEDGE's header and body are the followings.

```
header = VK_ACKNOWLEDGE


/*XDR*/
struct VKAcknowledge {
    void;
};
```

The following table shows addresses for sending and receiving.

header	send from	send to
VK_CONNECT	any	kernel (port 6000)
KV_CONNECT_OK	any	sender VK_CONNECT
KV_CONNECT_ERROR	any	sender of VK_CONNECT
VK_ACKNOWLEDGE	sender of VK_CONNECT	sender of KV_CONNECT_OK

6.4 connection to Agents

Agents require connection with *kernel* via port 6000

```
header=AK_CONNECT
```

Its body has the following format.

```
/*XDR*/
struct AKConnect {
    int version;        /* must be 0 */
    int temporaryId;    /* any value */
    int agentType;
};
```

TemporaryId's value can be set to any value by a corresponding agent module. agentType is the kind of agent that is connecting. The value is one of the followings.

agentType	agent's kind
1	Civilian
2	FireBrigade
4	FireStation
8	AmbulanceTeam
16	AmbulanceCenter
32	PoliceForce
64	PoliceOffice

A UDP packet for connection a civilian agent has the following fromat.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 20	header=0x20
4	FF FF FF FF	length of body
8	00 00 00 00	version=0
12	00 00 00 00	temporaryId=0
16	00 00 00 01	agentType=Civilian
20	00 00 00 00	header=HEADER_NULL

Kernel returns KA_CONNECT_OK when the connection succeeds, returns KA_CONNECT_ERROR in a case of failure. *Simulators* returns AK_ACKNOWLEDGE, it receives KA_CONNECT_OK.

Kernel sends KA_CONNECT_OK again, when it does not receive SK_ACKNOWLEDGE after a specified time. KA_CONNECT_OK's header and body are the followings.

```
header = KA_CONNECT_OK


/*XDR*/
struct KAConnectOk{
    int temporaryId; /* value handed over by AK_CONNECT */
    int id;          /* ID of itself */
    Object self;     /* Object controlled by this agent */
    Objects map;     /* information that Object self knows before disaster */
    /* if there is extra data added, they should be neglected. */
};
```

temporaryId is the value handed over by AK_CONNECT. id is an ID of the agent object. Kernel sends the agent Object map information that kernel creates from GIS data. Kernel sends it in a binary form [9] KA_CONNECT_ERROR's header and body are the followings.

```
header = KA_CONNECT_ERROR


/*XDR*/
struct KAConnectError {
    int temporaryId;    /* value handed over by AK_CONNECT */
    string reason<255>;
    /* if there is extra data added, they should be neglected. */
};
```

temporaryId is the value handed over by AK_CONNECT. The string *reason* shows why it fails.

AK_ACKNOWLEDGE's header and body are the followings.

```
header = AK_ACKNOWLEDGE


/*XDR*/
struct AKAcknowledge {
    int id;    /* ID of an Object that is controlled by this agent. */
};
```

id in AK_ACKNOWLEDG is the same value as id in KAConnectOk.

The following table shows addresses for sending and receiving.

header	send from	send to
AK_CONNECT	any, can share with other agents.	kernel (port 6000)
KA_CONNECT_OK	any	sender of AK_CONNECT
KA_CONNECT_ERROR	any	sender of AK_CONNECT
AK_ACKNOWLEDGE	sender of AK_CONNECT	sender of KA_CONNECT_OK

[9]Note: The value of version must be set 0. version = 1 is allowed only for kernel developers. When agent developers set verion 0, they should know (1) kernel omits data conversion and data transmission in order to make time required for connection short, (2) the usage 'version = 1' is not guranteed for future.

6.5 sending Sensory information to agents

Kernel sends KA_SENSE to agents every cycle.

```
header = KA_SENSE
```

Its body has the following format.

```
/*XDR*/
struct KASense {
    int id;             /* ID of itself */
    int time;           /* current time */
    Object self;        /* information of itself */
    Objects objects;    /* information of surroundings */
    /* if there is extra data added, they should be neglected. */
};
```

`id` shows the Object that should receive sensory information. `time` is the current time of simulated world. `self` is information on itself and `objects` is information on current surroundings.

The following table shows addresses for sending and receiving.

header	send from	send to
KA_SENSE	any	sender of AK_CONNECT

Agents must commands in reply to receiving KA_SENSE.

6.6 receiving commands from agents

command is a block with a header that is over 0x80 and under 0xFF. The first 4 byte of body is ID of object that a sender controls. The usage of commands are future topics. Basically, commands will be used for communication among modules and will be transfered to simulators. The following table shows addresses for sending and receiving.

header	send from	send to
0x80 ~ 0xFF	sender of AK_CONNECT	sender of KA_CONNECT_OK

6.7 command transfer to Simulators

Kernel gathers commands sent from agents into one block, and sends it to all simulators. KSCommands's header and body are the followings.

```
header=KS_COMMANDS
```

```
/*XDR*/
union CommandBodies(int senderId) {
case 0:
    void;
default:
    opaque unknown<>;
    CommandBodies next;
};

union PackedCommands(Header command) {  /* command */
case HEADER_NULL:
    void;
default:
    int remainder ToNextCommand;
    CommandBodies commandBodies;
    PackedCommands next;
};

struct KSCommands {
    int time;
    PackedCommands commands;
    /* if there is extra data added, they should be neglected. */
};
```

`time` is the current time of simulated world. `commands` is information of all commands that are sent by agents during one cycle. `commands` is divided into three components: header, the first four bytes of body, and the rest of body. They are assigned to command, senderId, unknown. Commands with the same header are unified into one command. remaindertoNextCommand is the length of commandBodies in byte.

When three commands (Fig. 4) are sent from agents, *kernel* sends a packet (Fig. 5).

The following table shows addresses for sending and receiving.

header	send from	send tp
KS_COMMAND	any	sender of SK_CONNECT

Agents must SK_UPDATE in reply to receiving KS_COMMANDS.

6.8 receiving updated information from simulators

Kernel receives updated information from simulators. Its body has the following format.

```
header=SK_UPDATE
```

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 80	header=0x80
4	00 00 00 10	length of body
8	00 00 01 00	sender id
12	00 00 01 01	
16	00 00 01 02	
20	00 00 01 03	
24	00 00 00 80	header=0x80
28	00 00 00 10	length of body
32	00 00 02 00	sender id
36	00 00 02 01	
40	00 00 02 02	
44	00 00 02 03	
48	00 00 00 81	header=0x81
52	00 00 00 10	length of body
56	00 00 03 00	sender id
60	00 00 03 01	
64	00 00 03 02	
68	00 00 03 03	
72	00 00 00 00	header=HEADER_NULL

Figure 4: three commands sent from agents

```
/*XDR*/
struct SKUpdate {
    Objects differences;
};
```

`differences` is the updated information from simulators. *Kernel* updated the properties of objects in the simulated world.

The following table shows addresses for sending and receiving.

header	send from	send to
SK_UPDATE	sender of SK_CONNECT	sender of KS_CONNECT_OK

6.9 managing properties of agents

Kernel decreases hp, increases stamina.

6.10 sending update data to simulators

Kernel sends to all simulators data for updating their simulation world. Its body has the following format.

OFFSET	HEX BYTES	COMMENTS
0	00 00 00 62	header=0x62
4	00 00 00 5C	length of body
8	00 00 00 10	time=16
12	00 00 00 80	command=0x80
16	00 00 00 2C	remaindertoNextCommand
20	00 00 01 00	senderId
24	00 00 00 0C	length of unknown=12
28	00 00 01 01	unknown
32	00 00 01 02	
36	00 00 01 03	
40	00 00 02 00	senderId
44	00 00 00 0C	length of unknown=12
48	00 00 02 01	unknown
52	00 00 02 02	
56	00 00 02 03	
60	00 00 00 00	senderId=0
64	00 00 00 81	command=0x81
68	00 00 00 18	remaindertoNextCommand
72	00 00 03 00	senderId
76	00 00 00 0C	length of unknown=12
80	00 00 03 01	unknown
84	00 00 03 02	
88	00 00 03 03	
92	00 00 00 00	senderId=0
96	00 00 00 00	command=0x00
100	00 00 00 00	header=HEADER_NULL

Figure 5: a unified packet sent from Kernel to Simulators

```
header=KS_UPDATE
```

```
/*XDR*/
struct KSUpdate {
    int time;
    Objects differences;
    /* if there is extra data added, they should be neglected. */
};
```

`time` is the current time of simulated world. `differences` is information for updating simulation world.

The following table shows addresses for sending and receiving.

header	send from	send to
KS_UPDATE	any	sender of SK_CONNECT

6.11 sending update data to viewers

Kernel sends to *viewers* updated data of the simulated world. Its body has the following format.

```
header=KV_UPDATE
```

```
/*XDR*/
struct KVUpdate {
    int time;
    Objects differences;
    /* if there is extra data added, they should be neglected. */
};
```

`time` is the current time of simulated world. `differences` is information for updating simulation world.

The following table shows addresses for sending and receiving.

header	send from	send to
KV_UPDATE	any	sender of VK_CONNECT

6.12 sending update data to GIS

Kernel sends to *GIS* updated data of the simulated world. Its body has the following format.

```
header=KG_UPDATE
```

```
/*XDR*/
struct KGUpdate {
    int time;
    Objects differences;
    /* if there is extra data added, they should be neglected. */
};
```

`time` is the current time of simulated world. `differences` is information for updating simulation world.

The following table shows addresses for sending and receiving.

header	send from	send to
KG_UPDATE	any	sender of GK_CONNECT_OK

6.13 time keeper

kernel advance the time one unit in the simulated world.

7 GIS Format

7.1 building.bin

This section explians GIS data form. This data format [10] is used for data echange among modules in Version. 0. Modules receive the data from Kernel when they connect at first, and make Building objects cited in 4.11.

At version.0, GIS data is based on the 19 standard coordinate (19_s). 19_s is the coordinate used in Japan Geographical Survey Institute. The unit in 19_s is meter, however, mili_meter is used in RoboCup-Rescue in order to make the data type is integer. The transformations from (x, y) coordinate to (x19, y19) in 19_s are

$$x19 = (y + A2)/1000, y19 = (x + A3)/1000.$$

data format:

	data type	contents	comment[unit]
A1	unsigned long	the number in 19_s	data of Kobe is 5 standard coordinate ∇ 畑山さん: 要説明
A2	long	offset value of 19_s (x coordinate)	[mm]
A3	long	offset value of 19_s (y coordinate)	[mm]
A4	unsigned long	total points number	the number of buildings (p)
A4-1 ∼ A4-p	−	information of each building	cf. record format described below

record format (data of a buildin)

[10] It is different from the data format used in GIS module.

	data type	contents	comment[unit]
R1	unsigned long	the size of this record	[*4 byte]
R2	unsigned long	ID number	−
R3	unsigned long	standard point (x coordinate)	[mm]
R4	unsigned long	standard point (y coordinate)	[mm]
R5	unsigned long	the number of floors	[floor]
R6	unsigned long	kind of building	0: it has floors less than 3. 1: more than 4 floors.
R7	unsigned long	ignition flag	The value is 1 for the building that Fire simulator sets fire to, otherwise 0.
R8	unsigned long	fieryness	initial value = 0
R9	unsigned long	brokenness	initial value = 0
R10	unsigned long	the number of connetion nodes	q (at present no input, set default 0)
R10-1 $\sim$ R10-q	unsigned long	the ID of node connecting the entry of this building	(at present, no input)
R11	unsigned long	buildingShapeID	the same as R2
R12	unsigned long	buildingAreaGround	[*0.01 m^2] cf. 4.11
R13	unsigned long	buildingAreatotatal	[*0.01 m^2] cf. 4.11
R14	unsigned long	buildingAttributes	0, 1, 2 cf. 4.11
R15	unsigned long	buildingApexes	r cf. 4.11
R15-1 $\sim$ R15-2r	unsigned long	coordinates of buildingApexes	[mm]odd number: x-corinate, even number: y-coordibate

8 install of prototype

The prototype version is available at URL: http://ne.cs.uec.ac.jp/˜koto/rescue. Expand the archived file under your specified directory. The followings are from README file of Prototype Ver.0.20.

8.1 Unix version

Requirements :

1. C++, socket library, Java Development Kit 1.1

How to run Make and execute sh files in RUN directory, from 0 to 5 in order. There are two batch files which are numbered 2. Click sun_viewer or ms_viewer, if you want to use SUN's JavaVM or MicroSoft's JavaVM.

8.2 Windows version

Requirements :

1. Windows 95/98/NT (including Winsocks2)..
2. Java Runtime Environment 1.1 (or corresponding Microsoft Java VM).

How to run Execute batch files in RUN folder, from 0 to 5 in order. There are two batch files which are numbered 2. Click sun_viewer or ms_viewer, if you want to use SUN's JavaVM or MicroSoft's JavaVM.

Source Code The source codes are open for Window version.

8.3 GIS data

In Data folder (directory), there are three subdirectories:

10 : 1/10 model of Nagata Words,

100 : 1/100 model,

1000 : 1/1000 model.

With replacing files, the simulated world will change its size.

8.4 Tuning

In config.txt, there are parameters to change the conditions of simulators. When pacakets are lost frequently, set send_udp_wait larger values.

8.5 LogFiles

Kernel outputs logfiles of execution by deleting # marks in config.txt.

9 Programmer's Guide

9.1 agent

This section describes how to use commands in section 3.4.1 when you program agents. (An example of source code will be open by Mr. Ohta at this April.)

move: *CIVILIAN*, *FIRE_COMPANY*, *AMBULANCE_TEAM*, *POLICE_FORCE* use this command to move and *Traffic simulator* takes care of it.

```
header=AK_MOVE
struct AKMove {
    u32 id;         // sender agent ID
    IDs route;      // a list of objects of Node, Building, etc. ,
                    // which the sender passes through.
};
```

say: All agents use this command to say with natural voice and *kernel* takes care of it.

```
header=AK_SAY
struct AKSay {
    u32 id;         // sender ID
    string message; // message
    ID target;      // ID of a target which is said by id.
};
```

tell: All agents use this command to say with communication medias such as telephone, wireless phone and *kernel* takes care of it.

```
header=AK_TELL
struct AKTell {
    u32 id;         // sender ID
    string message; // message
    ID target;      // ID of a target which is said by id.
};
```

extinguish: *FIRE_COMPANY* use this command to sprinkle water and *fire simulator* take care of it after *kernel* checks.

```
header=AK_EXTINGUISH
union KAExtinguishNozzles switch(ID target) {
case 0:
    void;
default:
    u32 direction;  // 0 degree is the direction of Y-axis and
    // to 1295999 minutes in counterclockwise direction.
    u32 positionX;  // the position of the nozzle,
    u32 positionY;  //     it is not so far from the car's position.
    u32 quantity;   // the amount of water from one nozzle:
                    //     1 unit = 0.001m*m*m/min
    KAExtinguishNozzles next;
    };
    struct KAExtinguish {
        u32 id;         // sender ID
        KAExtinguishNozzles nozzles;
    };
```

rescue: *AMBULANCE_TEAM* use this command to rescue buried people and *miscellaneous simulator* take care of it.

```
    header=AK_RESCUE
    struct AKRescue {
        u32 id;         // sender ID
        ID target;      // ID of a target which is rescued by id.
    };
```

load: *AMBULANCE_TEAM* use this command to load the injured into an ambulance and *miscellaneous simulator* take care of it.

```
    header=AK_LOAD
    struct AKLoad {
        u32 id;         // sender ID
        ID target;      // ID of a target which is loaded by id.
    };
```

unload: *AMBULANCE_TEAM* use this command to unload the injured from an ambulance and *miscellaneous simulator* take care of it.

```
    header=KA_UNLOAD
    struct AKUnload {
        u32 id;         // sender ID
                // No target, because only one ID is on the ambulance.
    };
```

unload: *POLICE_FORCE* use this command to restore the blockaded road so that cars can pass and *miscellaneous simulator* take care of it.

```
    header=AK_CLEAR
    struct AKClear {
        u32 id;         // sender ID
        ID target;      // ID of a target which is restored by id.
    };
```

The header has the following values.

```
AK_REST       = 0x80;  AK_MOVE       = 0x81;  AK_LOAD       = 0x82;
AK_UNLOAD     = 0x83;  AK_SAY        = 0x84;  AK_TELL       = 0x85;
AK_EXTINGUISH = 0x86;  AK_RESCUE     = 0x88;  AK_CLEAR      = 0x89;
```

10 FAQ

10.1 Architecture

1. Hi, I have two concerns:

   1)It says that all communication between agents goes through the kernel. Won't this become a bottleneck when the number of agents increase? Especially since the next step is to make each individual an agent and not a team as an agent as it is now

   2)Also kernel waits for a fixed time to get back the commands issued by the agent. It filters these and send them to the simluators. What about those commands that come in late? Also when the simulators return their results to the kernel it integrates these results and returns it to the GIS and to the simulators. Here to it waits for only a fixed period of time. What happens if the simulator's reply reaches the kernel after it has begun integrating the rsults from the other simulators. Won't a incorrect picture of the world be created and sent to both the gis and the individual agents? Thanks for your time.

   (Ranjit Nair, Jan. 2000)

   - Yes, we also have these concerns. These problems about distributed kernel and asyncronous simulaton will be central issues of the next kernel (version 1). Join us for discussion!

     (Ikuo Takeuchi)

   - For the short term plan, rescue simulation project will proceed.

     Phase 0 (— 2000.3)
     Feasibility study of a disaster-agent simulator for a very simple agents and environments.

     Phase 1 (— 2001.4)
     Simulator development of limited disaster and limited agents. The 1st research evaluation conference.

     Phase 2 (— 2005.4)
     Simulator development of larger-scale disaster simulator, heterogeneous agents.

     This manaul is a document at Phase 0. Designing the simulation at Phase 1 must take care of the problems. They invole your suggested points, Ikuo's points and, GIS format, viewer, time-management among simulators....

     (Tomoichi Takahashi)

10.2 Viewer

1. This is a snapshot of what I obtained on running the sample simulation(kernel-unix-0.17). Please confirm that this was what it was supposed to look like? I was expecting fancier graphics. Isn't the simulation supposed to be modelled on a city block in Kobe? Will future releases have this? When can we expect these releases.

   (Ranjit Nair, Jan. 2000)

Table 3: Objects in test environments [number (byte size)]

scale	1/1000	1/100	1/10	1/1
area size (m)	31	160	521	2217
static objects				
road	4(60)	125(1,875)	818(12,270)	9,776(146,540)
node	5(35)	119(833)	765(5,355)	9,143(64,001)
building	1(13)	99(1,287)	778(10,114)	9,357(121,641)
subtotal	10(108)	343(3,995)	2,361(27,739)	28,276(332,282)
autonomous agents				
Civilian	1(8)	8(64)	76(608)	934(7,472)
Ambulance Team	1(8)	2(16)	5(40)	5(40)
Fire Brigade	1(10)	2(20)	10(100)	10(100)
Police Force	1(8)	2(16)	10(80)	10(80)
subtotal	4(34)	14(116)	101(828)	959(7,692)
total sum	14(142)	357(4,111)	2,462(28,567)	29,235(339,974)

- The snapshot is correct. In kernel-unix-0.17, the kernel is official, but the other modules are just sample to test the kernel. We released version 0.19. It contains a semi-official GIS and displays a small area in Kobe. But it has a bug about endians and needs little endian architecture machine.

  (Tetsuhiko Koto)

10.3 Performance

1. Where can I get test Environment?

   - There are four test GIS data of different scales. at URL:

     http://ne.cs.uec.ac.jp/~koto/rescue. The GIS data are Nagata area with 1/1000, 1/100, 1/10 and 1/1 scale (Table 1).

     Figure 6 shows the 2-D viewer of 1/10 model. The entries in the first row is the length of displayed area. The edge of displayed area is 521m. The second group of rows in Table 1 are static objects, and the third group is autonomous agents.

     The green spots are buildings, the yellow spots show the ignition pointes and the white lines are roads. The color of buildings turns to red when the building burns, and turns to blue when fire brigades extinguish the fires. The points on the lines represent autonomous agents.

     The numbers in the entry are the numbers of objects, and the byte size (in parentheses). (Tomoichi Takahashi)

2. How much machine power is necessary to run test environment?

   - I don't know exactly. Table 4 shows the machine specifications that we tested at 3/5/2000. We used seven machines connected through 100M network. (Tomoichi Takahashi)

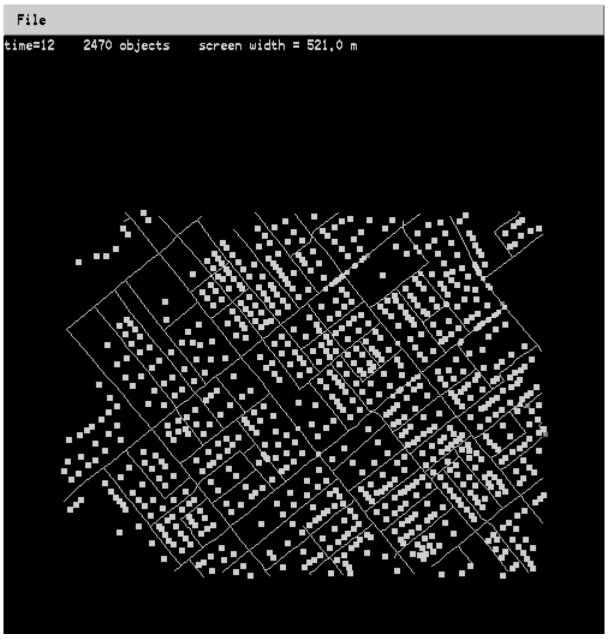

Figure 6: 2D viewer of 1/10 scale.

3. What about the test ? Did it work smoothly?
   - At 1/1000, 1/100, 1/10 scale, it worked as we expected. At 1/1 scale, it did not work. The followings are the reasons:
     (a) At the initialization phase, every module gets the GIS data. The GIS data of 1/1 scale is 330KB, and is sent via UDP/IP. This causes packet lost during receiving them. And as the number of modules plugged in, the total transmission time and the number of lost packets increase. And the time for initialization becomes more than 30 minutes.
     (b) During simulation, kernel sends the changes of circumstances to agents by calculating within the limited area. At present, the cost of this calculation is proportional to the number of agents. The increase from 76 agents to 934 agents is hard for the present kernel.
   - For the above problems, the followings are commented:
     (a) As far as the initial step, the transmission of GIS data should be via TCP instead of UDP in order not to miss packets. (Tadokoro, Aida, March/2000)
     (b) The modules are divided into two categories. The one category is disaster simulator, and the other is agent. The disaster simulators change the world itself under the complete data, while the agents

Table 4: machine environments for test

	components	CPU	Memory	OS
1	kernel	P3-600x2	512	WindowsNT/FreeBSD
2	GIS	P3-733	512	WindowsNT
Simulator				
3	Fire	P3-733	512	WindowsNT
4	Road blockade	P3-733	512	WindowsNT
5	Building blockade	P3-733	512	WindowsNT
6	Traffic	P3-733	512	Turbo Linux
Agents				
7	Civilian	P3-600	512	FreeBSD
	Fire			Linux
	Ambulance			Linux

move in the world with incomplete data. The number of disaster simulators is fixed, and the number of agents is variable.
In order to keep the cost in spite of the increase of the agent's number, there is a proxy between kernel and agents. (Ituki Noda, March/2000)

11 Discussion toward Version. 1

(

11.1 Architecture

1. Time management: fixed RTK [11] or flexible RTK? (Yoshitaka Kuwata, 2000/2/13) At version0, one simulation step corresponds to one minute in the simulated world. 4,320 = 60 * 24 * 3 simulation steps become 3 days. Agents or simulators calculate according to their own models, and the required times are different from each other. For example, one minute is good for fire simulations, while it is too long for traffic simulations. Because a car moves more than one block in one minute. RTK(60) may make impossible to transfer protocols among agents and simulators within one minute. It causes the simulation is out of the real one.

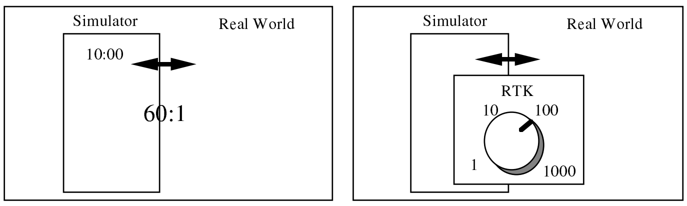

Figure 7: Real-Time Knob(RTK)

2. is calculation guaranteed to complete ? (Yoshitaka Kuwata, 2000/2/13)

   (a) why guarantee to complete within one step is necessary?
   Under RTK(60) system, calculation of both agents and simulators are expected to finish within one step. Agents in RoboCup soccer simulation leagues are soccer players. In the soccer games, whether the players run after deliberation, dash for the ball at first, or else are the problem of soccer agents themselves. This situation applies also to agents of Rescue project simulation.

   However, disaster simulations in rescue simulation should complete its calculation in one simulation step to keep the simulated world consistent. If a fire simulator does not finish in one step, the simulated fire does not spread out. Is it possible for disaster simulators to complete their calculation in one simulation step when the simulated world becomes larger than the assumed world at version 0 ?

   Incremental algorithm or anytime algorithm are proposed for real time simulations. Applying these algorithms to simulators that have been developed already is difficult. At least, some efforts including the computer

[11]The ratio of simulation time and real time is referred to Real-Time Knob (RTK) after this. Version 0 is RTK(60) system, it means that the ration is 60.

power up should be made that the modules of disaster simulation return the result.

(b) real-time / non real-time simulators ? (Yoshitaka Kuwata, 2000/2/13) Real-time disaster simulators are important. On the other hand, logistics planning or city-reconstruction planning by government offices require precision of simulation rather than its speed.

In future, the non real-time simulators will be plugged in. The rescue simulation should support interfaces for both real-time and non real-time simulators.

Question Is it possible for kernel to control the RTK of simulators? If so, given a table where the granularity of simulators are listed, the kernel indicates RTK's value to simulators. When a simulator fails to return its results in one simulation step, the value of RTK will be set larger one. (Kento Aida, 2/15)

Answer As mechanism of simulators, I think it is possible to control the RTK, although programming simulator takes efforts. For example, see Boddy, M., & Dean, T. "An analysis of time-dependent planning.", in proceedings of the sixth national conference on Artificial Intelligence(AAAI) , pp. 49-54, 1988 or Boddy, M. & Dean, T. "Solving time-dependent planning problem", in eleventh international joint conference on Artificial Intelligence (IJCAI), pp. 979-984, 1989.

When more people enter the community of rescue project, I think it is a good policy to make their existing simulation programs plugged in easily. So, the realistic and simple way is that kernel or some module will manage the characteristics of the newly plug in simulator, isn't it ? But it does not solve our problem, because we must set all RTK to the highest value. (Yoshitaka Kuwata, 2/15)

3. Communication cost, performance of kernel(Yoshitaka Kuwata, 2000/2/13) The communication size of GIS data and the amount of kernel computation will be bottleneck of execution. (Ranjit's comment in page 48.) The followings will be considered:

   - distribution of GIS functionally and/or regionally,
   - distribution of kernel module,
   - event driven type kernel. (Takeuchi's commnet in page 55, Kuwata's comment in page 57).

Question One of distribution ways is to divide them into domains like DNS. There will be one kernel and one GIS within one domain. In this case, the division that makes dependence among domains little is desired. (Kento Aida, 2/15)

Answer I agree with your suggestion basically, however, there are many kinds of information linked over domains. (The followings are my guess:) When a fire occurs at the boundary of domains, a fire simulation must send data to both kernels of domains. In a domain, there is usually one kernel. When fires occur at the domain, the kernel divides the domain dynamically into smaller ranges and forks other kernels to manage the small ranges. This is an interesting research theme, but this seems to be difficult. Anyway, more detailed analysis will be needed. (Yoshitaka Kuwata, 2/15)

Question Next idea is to use simulators more than one that simulate the disaster of the same kind in a domain. By using some simulators, when the load of simulating one disaster, for example fire, in one domain, the load will be distributed to each simulator. By this method, distributed simulation inter-domains and intra-domain will be done. (Kento Aida, 2/15)

Answer In HLA(High Level Architecture), Federation architecture defines dynamic hierarchical structure. (Can anyone follow ?) (Yoshitaka Kuwata, 2/15)

Question The other idea is to assign simulators flexibly according to the demand of simulation power. One kernel of a domain that thinks the power of simulation insufficient asks kernels of other domains to support it. (Kento Aida, 2/15)

Answer For that, a meta-controller that manages kernels is required. Implementation using CORBA is a practical solution, isn't it? (Yoshitaka Kuwata, 2/15)

Question The kernel will do the following functions to realize the above functions.

- synchronization and communication among domains,
- segmentation of a domain and allocation of simulators.

The kernel needs the data-sheet on the speed of machine-computation, the amount of simulation-time, etc. Adding these functions makes the kernel big, so it may be good that adding functions are separated from kernel.

```
+--------+   +---------+
| kernel |---|scheduler|
+--------+   +---------+
    ||      /
  +-----+  /
  | GIS |--
  +-----+
```

(Kento Aida, 2/15)

Answer This figure is a similar to my figure (cf. Fig. 8). The scheduler in the above figure manages kernel tasks, while the scheduler in my figure controls RTK. (Yoshitaka Kuwata, 2/15)

11.2 GIS

1. More GIS ? (Yoshitaka Kuwata, 2000/2/13)
At version 0, the GIS module maintains geographical data as a whole, and other modules acess the data via the kernel. At future versions, there will be requests

to make use of GIS data at more detailed levels or from different points. For satisfying these requests, GIS data will increase in size and types. Distributed GIS modules must be considered for the increase of data. Of course, distribution of data will need consistency among them.

2. Management of agents such as civilian, cars (Yoshitaka Kuwata, 2000/2/13)
   The GIS module keeps record of not only static objects such as buildings but also civilian agents, car agents that move. Management of objects that move or don't move should be separated from computation efficiency. This separate management of building objects and civilian will be useful when several GIS modules will be used.

11.3 Utilities / Tools

1. Snapshot, rollback, rerun:
   The following functions are desired to debug programs, to analyze behaviors:

   Snapshot / Rollback : save all data on agents, simulations, GIS data, and kernel status at a specified time point.

   Rerun : restore the save data and start the rescue simulation from the point. In this case, persistence of objects should be managed. (2) (Yoshitaka Kuwata, 2000/2/13)

2. interoperabilty:
   Rescue project simulation is executed on computers connected to network. Protocol data format and data structure are desired to be independent of programming languages and machines.

   Compatibility is desirable for version upward. (Yoshitaka Kuwata, 2000/2/13)

11.4 performace

1. network
   At the interface test at Feb.13, the simulation environment (computers and network) seems to work as hard as possible even to simulate 1/10 model of Nagata ward. I think a network traffic team necessary to check what kind of packets are on the network. (Ikuo Takeuchi, 2/14)

2. kernel load:
   Think a fire occurs at a mesh. Is it sufficient to notify the fire to agents in the neighbor meshs ? The fire event should be announced more widely than 8m, especially for fire agents, vision range is insufficient for fire agents. The points that I think more important than the previous one are (1) the load in kernel seems to be high, and (2) fire agent can see only 8m ahead.

   At present, Kernel divides the simulated world into meshes with 10m grid, and calculates eight neighbor meshes around each agent at every step. It is better to program that the changes in the world should be notified from agents. Then there is no need for kernel to check eight neighbor meshes around an agent that says nothing. (Ikuo Takeuchi, 2/14)

3. event driven programming style is better than object based programming style :

   When several fires occur in a mesh, the events are sent respectively or together ? The events occurred in a mesh should be unified and selected according to the world model of kernel. Kernel will send the information to agents in farther mesh according to the model. For example, explosion event will be announced simultaneously to agent in wide range, while moving cars may be within a few meshes.

   In RoboCup rescue simulation where at most fires occurs at 10,000 points (the number of buildings), the event driven programming style requires less computation than agents based programming style by that a lot of agents look around every time. (Ikuo Takeuchi, 2/14)

12 Proposals to Version 1

12.1 Architecture

1. management of simulation time & separation of objects management: (Yoshitaka Kuwata, 2000/2/13)
   Kernel is divided into two modules, RT-scheduler and Command(Protocol)-Interpreter to reduce the amount of communication.

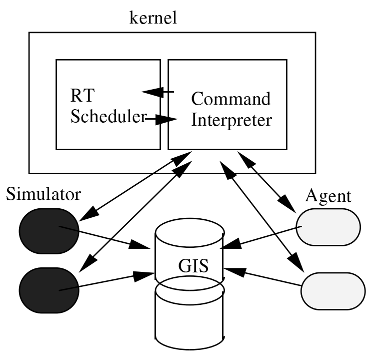

Figure 8: Division kernel's role into two modiles

   - Command(Protocol)-Interpreter
     Command-Interpreter will handle communication between agents and modules.
   - RT-scheduler
     Kernel will pass message to the GIS module with event driven scheduler. The basic flow is shown in fig 10.

     Event-Queue will be manipulated to gurantee the simulation,

     (a) limit the number of requests from agents,
     (b) give priority of events,
     (c) cut down the evens.

2. Management of distributed objects (Yoshitaka Kuwata, 2000/2/13)
   CORBA?

References

[1] Hiroaki Kitano, Satoshi Tadokoro et al., RoboCup-Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research, Proc. IEEE SMC, 1999.

[2] Satoshi Tadokoro, Hiroaki Kitano et al., The RoboCup-Rescue concept, The RoboCup-Rescue Committee, 1999.

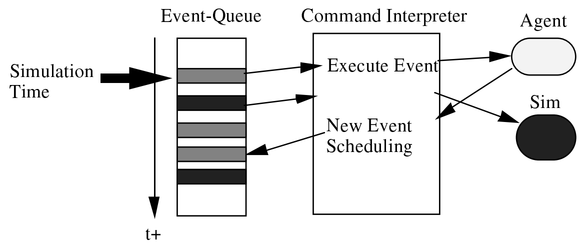

Figure 9: Model of event-driven simulation

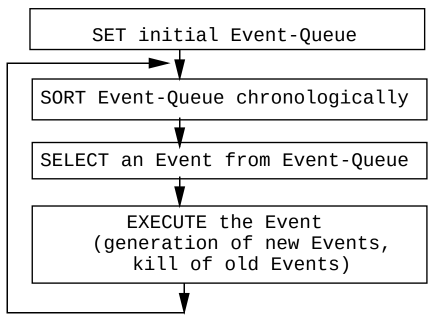
SET initial Event-Queue
SORT Event-Queue chronologically
SELECT an Event from Event-Queue
EXECUTE the Event (generation of new Events, kill of old Events)

Figure 10: Basic handling of event-queue

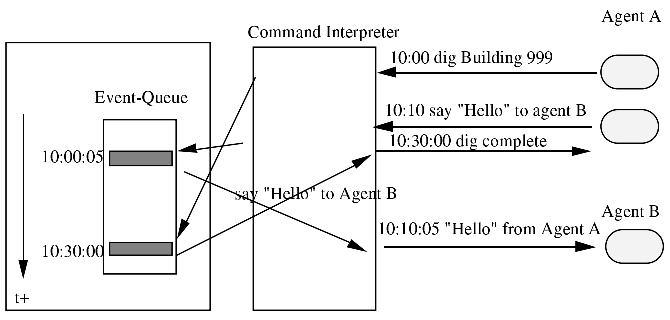

Figure 11: Communicaiton between agents via. Command Interpreter