Phoenix Rescue Team: An Agent Paradigm and Implementation

Zijian Ren
zijian.ren@gmail.com

1. Introduction

Agents are autonomous entities which can sense from and act on environment. An agent could represent a software program, a robotic control system or an entire robot. We propose a general agent paradigm and implement our Phoenix rescue agents with this paradigm. This general agent paradigm is not only applied for rescue domain, but also for other domains such as foraging and exploration.

In following paragraphs, we first describe this agent paradigm and the formal agent design method. Then, we describe our implementation in communication, knowledge base, and action control. Conclusion is presented in the last section.

2. A General Agent Paradigm and Design

The agent paradigm consists of five major components: Sensing (S), Control Unit (CU), Knowledge Base (KB), Action (A) and Communication (C) in figure 1. The control unit can be roughly divided into three modules: action control, knowledge acquisition and usage, and communication control. The communication component includes two parts: messages from other agents and messages to other agents. The knowledge base is the most complex part in the whole paradigm.

Besides the above modules, agents also have goals and constrictions.

- Goals: To pick up more objects (foraging task), or rescue more humanoids (rescue task), or higher score (soccer task), or others.
- Constrictions: computational abilities, storage space, communication loads, physical action limitations[1] such as action range and energy consumed, and errors in communication and action etc.

[1] Physical action limitations are common in real robots but less common in robot simulations.

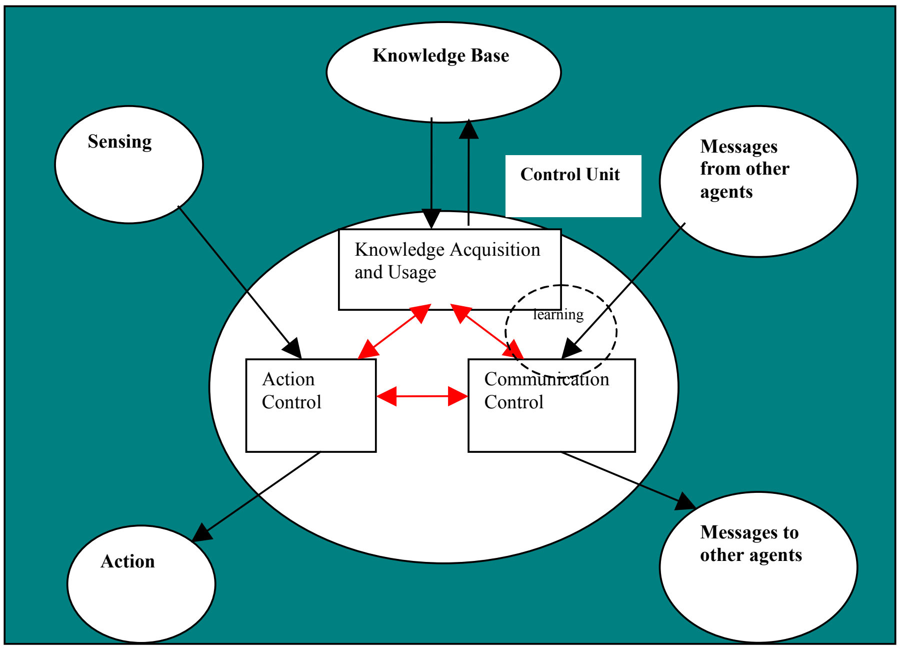

Fig. 1. An Agent Paradigm with Communication and Knowledge

- **Agent Categories**

With this agent paradigm, agents are classified by sensing, action, knowledge, and communication. There are ten agent categories in total (table 1):

- S and A: Non-communicated reactive agent
- S, A, and KB: Non-communicated cognitive agent
- S and C: Communication agent
- S, C and KB: Intelligent communication agent
- C: Blind communication agent
- C and KB: Blind intelligent communication agent
- C and A: Blind controlled agent
- C, A and KB: Blind intelligent controlled agent
- S, C and A: Communicated reactive agent
- S, C, A and KB: Communicated cognitive agent

In the rescue domain, three rescue agents (FireBrigade, AmbulanceTeam, and PoliceForce) could be a type of non-communicated reactive agent, non-communicated cognitive agent, communicated reactive agent, or communicated cognitive agent. Three rescue centers (FireStation, AmbulanceCenter, and PoliceOffice) could be a type of blind communication agent or blind intelligent communication agent.

Table 1. Agent Categories

	Sensing (S)	Action (A)	Communication(C)	Knowledge Base(KB)	Comments
Non-communicated Reactive Agent	●	●			standalone agent
Non-communicated Cognitive Agent	●	●		●	
Communication Agent	●		●		messages as actions
Intelligent Communication Agent	●		●	●	
Blind Communication Agent			●		message center
Blind Intelligent Communication Agent			●	●	
Blind Controlled Agent		●	●		controlled by other agents
Blind Intelligent Controlled Agent		●	●	●	
Communicated Reactive Agent	●	●	●		
Communicated Cognitive Agent	●	●	●	●	Fully functional agent

- **Formal Methods for Agent Design**

An agent can be represented as a tuple <S, A, C, KB, Constraint> according to the above agent paradigm, where S is sensing, A is acting, C is communication, and KB is a knowledge base.

The general problem is described as: Designing agents to optimize the performance P in a domain D.
Usually, S, A, and Constraint can be defined for this domain D. Then the performance P becomes the function of C, KB: P=perf(C, KB). The general problem is reduced to find appropriate ~~C~~ and ~~KB~~ in order to optimize P.

We propose a general formal approach for agent development:

1. Define S, A, and Constraint in the problem domain D. Determine agent types for this domain. For example, in some simple domains, C and KB may be unnecessary.
2. Set initial C and KB with constraints.
3. Get performance P by experiments. If P is good enough, current ~~C~~ and ~~KB~~ are appropriate and **STOP**; Otherwise, goto step 4.
4. Adjust C and KB. There are two adjustment methods: manually with human intelligence or autonomously by machine (computer) learning. Then go back step 3.

3. Communication Implementation

Currently, the simulation system restricts the number of message sent/received in each cycle for one agent, 4 messages for rescue agents (fire brigades, ambulance teams, and police forces) and 2*n for rescue center (fire station, ambulance center, and police office) which n is the number of rescue agents of the same kind as center. The maximum length for one message is 256 bytes.

To handle these limitations, we have several design principles:

1. Message is preferably human readable texts.
2. Only communicate important information, not all information, due to the length limitation and simulate real situation.
3. Message should be categorized so that receiving agents pick up the relevant messages and discard irrelevant ones.

The final communication scheme comes from several trial-and-error rounds.

- **Message Format**

The message format sent by rescue agents is "Time *t* Command *id* Degree *n*".
The message format between rescue centers is "Time *t* Command Degree d_1 id_{11} ... id_{1n}……Degree d_k id_{k1}…id_{km}".

Time stamp t shows the cycle when this message is sent, which will be an import communication factor in our thought. Currently, we just simply use this time stamp to discard obsolete messages. ID is the identifier of a building or a road.

Commands are designed with experiments:

- extinguish: report the burning buildings.
- putout: report the putout buildings.
- locate: report the buildings with injured humanoids.
- empty: report the building with all injured humanoids are rescued.
- clear: report the blocked roads.
- remove: report the roads with blockades are cleared.

The extinguish/putout pair is for fire brigade agents, locate/empty pair is for ambulance team agents, and clear/remove is for police force agents. The former in each pair is to transfer important knowledge so that rescue agents can make rational actions with this knowledge. The latter in each pair is to get rid of obsolete knowledge so that rescue agents don't waste time.

Degree is arbitrarily from 1 to 5 with the higher number represents more urgent. For example, normally, a burning building with degree 5 has priority to a burning building with degree 1 in fire brigade agents' viewpoints.

Complying with message number limitation, rescue centers packs similar short messages from rescue agents to several long messages. For example, a fire center receives three messages from fire brigades: "time 10 clear 128 degree 1", "time 10 clear 2257 degree 5", "time 10 clear 678 degree 1" (road 128 and 678 with blockade degree 1, road 2257 with blockade degree 5). The fire center packs these short messages into one long message: "clear degree 5 2257 degree 1 128 678". The final message from the fire center to the police office is "time 12 clear degree 5 2257 degree 1 128 678".

- **Future improvements:**

Time stamp should have more important role in future communication developments. More command categories if necessary.

4. Knowledge Base Implementation

Rescue agents are the type of communicated cognitive agent and rescue centers are the type of blind intelligent communication agent. Thus, there are two different knowledge bases for these two agent types.

- **Knowledge Base for Rescue Centers**

The goal of rescue center is to effective transfer useful information with restrictions. In software implementation, we design a "*Java class MessageBase*". The structure is:

Type	Identifier	Comment
Boolean	sent	Check whether this message is sent
int	receiveTime	the receiving time stamp
int	sendTime	the sending time stamp
int	degree	1 to 5, the higher, the serious
string	comment	reserved for future development

Knowledge base for fire station consists of four types of knowledge: buringBuildings, ***putoutBuildings***, blockadeRoads, and buriedBuildings.
Knowledge base for ambulance center consists of four types of knowledge: buriedBuildings, ***emptyBuildings***, buringBuildings, and blockadeRoads.

Knowledge base for police office consists of four types of knowledge: blockadeRoads, ***clearRoads***, buriedBuildings, and buringBuildings.

The knowledge type with a bold and italic item is special for one center, while the other three types are common for all centers.

Every type has similar structure "*Java class HashMap*" in software implementation. For example, knowledge "clearRoads" contains

road id1	message base 1 (class MessageBase)
road id2	message base 2 (class MessageBase)
…	…
road idn	message base n (class MessageBase)

In current implementation, a message can be added to the knowledge base but can't be removed. If a message is sent/obsolete, the sent flag of this message just be set to "Ture". The main purpose to keep all messages is for debugging.

In future, if the knowledge base is too huge to handle, remove or purge actions in knowledge base are options.

- **Knowledge Base for Rescue Agents**

We use several "*Java class HashSet*" for rescue agent's knowledge base such as burningBuildings, buriedBuildings, blockadeRoads.

There is an evolutionary process in design these knowledge types. Initially, only one type blockadeRoads is used for knowledge about blocked roads. Further, there is a need to separate this knowledge into two types: urgentRoads and normalRoads. Finally, with more classification, this knowledge is divided into six classes with degree information.

- degree 5: the most urgent blockade roads, need to be cleared immediately.
- …
- degree 1:the least urgent blockade roads, to be cleared only after there is no more urgent blockade roads.
- degree 0: unblocked roads.

Subsequently, the communication format is modified to satisfy this change as shown above "Time *t* Command *id* Degree *n*" and "Time *t* Command Degree d1 *id-1 … id-n*……Degree *dm id-x…id-z*". Initially, the communication format doesn't include "degree" segment.

However, due to limited time and human power, not all rescue agents use knowledge base with "degree". Our message processing is downward compatible to handle messages with/without "degree" segments.

A rescue agent acquires knowledge by communication and learning. Because online learning seems ineffective in current competition rule and setting, we don't use online learning methods such as reinforcement learning. There are debates over offline learning due to the difference of simulation and real world. Also, limited time and human power (one person) prevent us from some offline learning. Thus, in our current implementation, learning is not used.

- **Future improvements:**

In future, all knowledge base will be unified with “degree” information. However, downward compatible is still an important issue for software or hardware design.
More knowledge components can be added such as proper parameters, rules. Knowledge design is the most versatile and crucial part, also the hardest among all agent design parts.

5. Action Control Implementation

With knowledge in an agent’s hand, the rescue agent should make rational actions as much as possible to reach the goal. It becomes to design a good action control system.

Our action control systems are finite state machine (FSM) in the nature. In each state, the control system receives sensing information and knowledge from the knowledge base and makes an appropriate action.

- **Fire Brigade Agent**

We design four states: SEARCH, EXTINGUISH, REFILL, and GOTARGETS.
In “search” state, the agent tries to find fires or choose a target fire to extinguish if it has fire knowledge in its knowledge base. A general policy to choose a fire target is the fire degree. The higher a fire degree, the more priority it is chosen.

In “extinguish” state, the agent is extinguishing a target fire. The agent will continue to extinguish the same fire until the fire is put out. Then if there are nearby fires, put out those fires. Otherwise, search a new fire with new “search” state.

In “refill” state, the agent must go to a refuge (water center) to refill its water tank or is refilling at the center. Only when its water lever exceeds the require quantity, a preset parameter, this agent change to new states with sensing and knowledge.

In “gotargets” state, the agent already chooses fire targets and is moving to one of them, normally the nearest one with its routing algorithm. The agent may change its targets if it gets more urgent fire information.

- **Ambulance Team Agent**

We design four states: SEARCH, RESCUE, CARRY, and URGENT.
In “search” state, the agent tries to find buildings with buried humanoids or choose a target building if it has buried knowledge in its knowledge base.

In “rescue” state, the agent is rescuing an injured person. The state can’t be changed until the person is fully rescued (the buried degree is 0). Then, the agent state will change to “carry” state. If there are multiple ambulance agents who are rescuing the same person, only one ambulance team changes to “carry” state.

In “urgent” state, the agent is rescuing an injured rescue agent such as a fire brigade or a police force. Because these rescue agents will further do more rescue work than civilians, rescuing these agents has a priority.

In “carry” state, the agent is carrying an injured person and moving to a refugee. Once they reach the refugee, the rescue agent will change to a new state.

- **Police Force Agent**

We design three states: SEARCH, CLEAR, and GOTARGETS.
In “search” state, the agent tries to find blocked roads or choose a target road to clear if it has road knowledge in its knowledge base. A general policy to choose a road target is the road blocked degree. The higher a road locked degree, the more priority it is chosen.

In “clear” state, the agent is clearing a blocked road. This state can’t be changed until the road is cleared.

In “gotargets” state, the agent already chooses road targets and is moving to one of them, normally the nearest one with its routing algorithm. The agent may change its targets if it gets more urgent road information.

- **Miscellaneous**

Besides above special design for each agent type, there are several common action control strategies. In traffic jam situation, an agent moves to a safe building nearby if it can’t move in previous cycle. If an agent is in a burning building, it must moves out.

- **Future Improvements:**

Currently our strategy doesn’t do extensive calculations. In future, we may use other teams’ encircle methods to effectively extinguish a fire cluster.

6. Summary

Phoenix team is fully developed by Zijian Ren at University of Iowa. It attended the 2004 RoboCup Rescue Simulation Competition hold at Lisbon, Portugal by remote entry with YowAI2004 team’s help, and ranked 12th in the final.

The general agent paradigm along with the agent design method will be right direction for the (rescue) agent development. Several improvements are stated above. Due to the limited time and human power, these ideas are not implemented for potentially better performance.

Debugging is a crucial, yet difficult issue for multi-process simulation. We use console outputs and graphical Kuwata viewer for debugging. We hope there are more powerful graphic debug tools for multi-process simulation, similar to computer games.