Language Design for Rescue Agents

Itsuki Noda[1], Tomoichi Takahashi[2], Shuji Morita[3], Tetsuhiko Koto[4], and Satoshi Tadokoro[3]

[1] AIST & PRESTO
[2] Chubu Univ.
[3] Kobe Univ.
[4] Univ. of Electro-Communications

Abstract. We are proposing a model of communication and a specification of language for civilian agents in RoboCup Rescue Simulation. Robust information systems are critical infrastructure for rescue in huge disasters. The main issue of the information system in the rescue domain is that the most of information sources are civilians who may report incomplete information. Proposed model and specification is designed to reflect natural features of human communication behaviors. The model is also designed in order to provide for researchers to implement new design of information devices in the simulation flexibly. Using the model and specification, we can evaluate information systems and behaviors of rescue specialist agents.

1 Introduction

Communication of agents in the rescue domain is an important factor to affect the performance of rescue activities. Especially communication among civilians, who are the majority of agents in damaged area, will be the primary information source for rescue activities.

We are designing agent simulator modules for the RoboCup rescue simulator. The main purpose of the design is to provide a standard of communication among agents, especially civilians.

In this article, we describe a brief overview of the rescue simulator system (Section 2), and propose a layered model of agent communication (Section 3) and specification of a communication language for civilian agents (Section 4).

2 RoboCup Rescue Simulation System

Here, we like to provide brief overview of the rescue simulator (version 1) that we are designing.

Similar to the version 0 simulator[TTK$^+$00] and FUSS[NOD00], the system consists of the kernel and plug-in modules (Figure 1). The kernel provides functions to synchronize distributed expert modules and to manage shared data among modules. The plug-in modules are classified into three types: expert simulation modules, human-interface modules, and agent simulation/proxy modules.

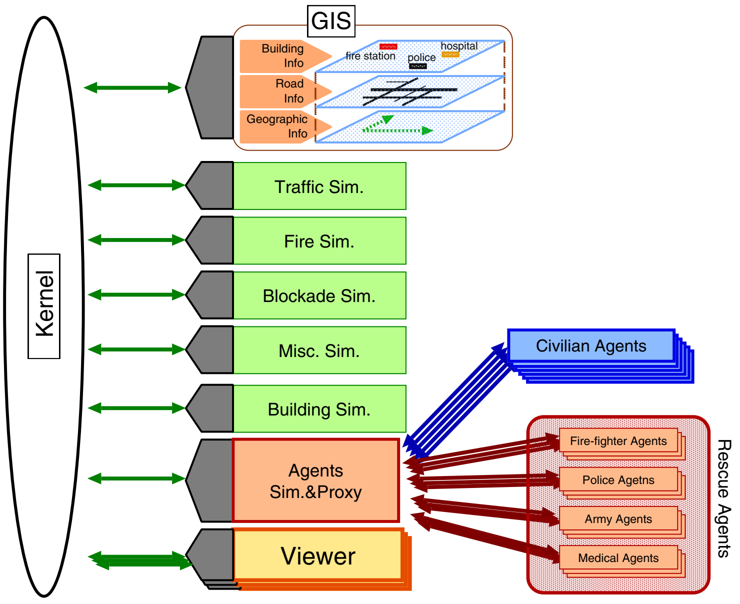

Fig. 1. Rescue Simulation System

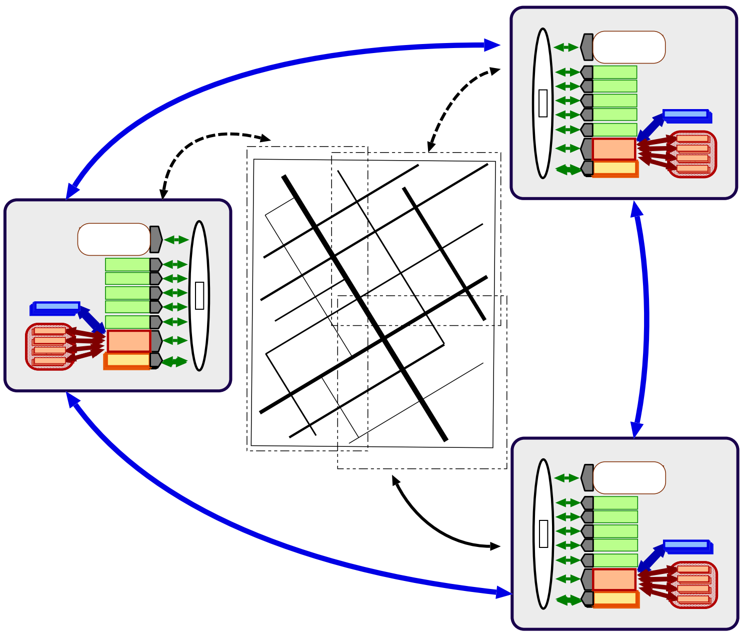

Fig. 2. Parallel Simulation with Overlapping

Each expert simulation module calculates changes of a phenomenon from an aspect like file, traffic, and so on. The human-interface modules provide a facility to generate real-time 2D and 3D visual image, and natural language description of the process of simulated phenomena. The agent simulation/proxy module performs two functions: (1) it simulates status of agents' bodies. (2) it communicates with agent-clients that become brains of the agents.

The system also has a facility to enable parallel simulation of divided areas as shown in Figure 2.

3 Agent Communication Model

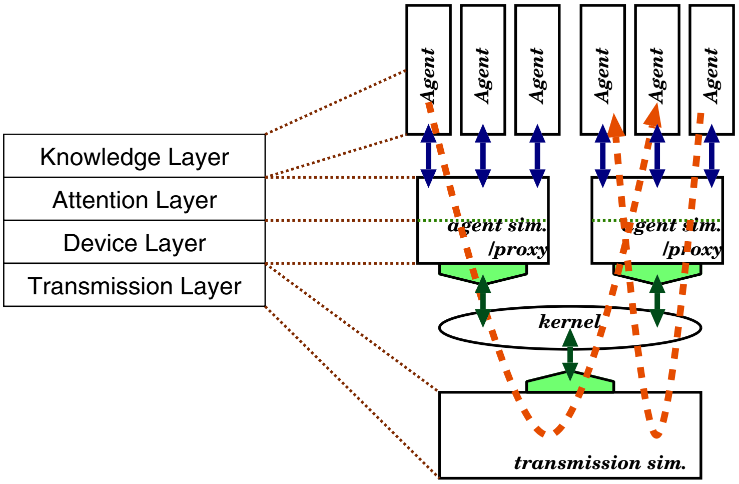

Fig. 3. Four Layers Model of Communication and Mapping to The Simulation System

3.1 Layered Commutation Model

In the rescue simulation, modeling agents' activities is a key issue. Especially, communication among agents is an important factor to decide the agents' physical performance.

In order to model the communication, we design *four layers model of communication*. The four layers are:

– **knowledge layer**
  to deal with syntax and semantics of communication. This layer is open for agent programmers.

- **attention layer**
  to simulate responsibility of agents' communication. For example, reading bulletin boards require agents' active attention. Even if information is on a bulletin board, an agent can not know the information without giving his/her attention to the bulletin board. On the other hand, a siren attracts attention of most of agents' even if they do not pay their attentions to the sound. This layer also controls the capacities of agents' communication.
- **device layer**
  to simulate availability and functionality of devices like telephone, PDA, PC, and so on.
- **transmission layer**
  to simulate transmission of physical signals like sound, electric wave, telephone line, and so on.

Figure 3 shows a mapping from the four layers to the simulation systems. The knowledge layer correspond to agent clients, in which original messages are generated and interpreted. The attention layer is in the agent simulator/proxy modules, which filter received information according to status of agents' attention and resources of agents' body. For example, when an agent is not paying attention to a bulletin board, it can not get information from the memo on the board. When an agent is listening a message, it can not listen another messages in the same time.

The device layer is also in the agent simulator/proxy modules, which filter information according to devices required in a specified communication method. For example, an agent need to have a mobile phone or find a wired phone to call a fire department office. The transmission layer corresponds to an expert simulator for the transmission. In the case of mobile phone, the simulator checks resource for the connection. When agents communicate by direct conversation, the simulator calculates the range the voice can reach.

The separation to the four layers helps for researchers to introduce a new method of communication into the simulation system. When researchers want to introduce their new PDA device designed for rescue into the simulator to test their facility for rescue domain, they should only develop their own device layer and combine it into the agent simulator/proxy modules. If they want to introduce new transmission method (like bluetooth) with PDA, it is possible by implementing both of device and transmission layers.

4 Agent Language for Civilian

Basically, specification of language used in the knowledge layer is open in the rescue simulation system. Agents can transfer information by any kind of formats with each other. However, it will be required to provides interoperability among heterogeneous types of agents that are developed separately. Therefore, we need a specification of the rescue agent language. For the first step, we are designing a language for civilian.

FIPA already specify languages for several applications. For example, FIPA KIF [FIP00c] and FIPA CCL [FIP00a, WCF+00]are designed respectively for interchange of knowledge and for information gathering. But no proposed specifications in FIPA fits for our purpose in the rescue simulation, because the language needs to reflect natural language features. Especially, civilian agents should behave as standard human civilians in the virtual world, so that civilian's conversation will include incomplete and ambiguous expression.

The natural language itself, on the other hand, is too wide and complex to be handled in the rescue simulation. There are many unsolved issues in the researches on natural language processing. Fortunately, we can limit variation of utterances for the rescue domain, where there are few complicated language phenomena.

Based on these consideration, we design a formal language.

4.1 Basic Syntax

Basically we use a similar syntax as S-expression of FIPA's communication language, that is:

```
(SpeachAct :sender Agent
           :receiver Agent
           :content Content
           ...)
.
```

Currently, the variations of *SpeachAct* are `inform`, `query-if`, `query-ref`, `request`, `commit`, `acknowledge`, and `not-understood`. An `inform` clause tells that the sender believe the *Content* is true. A `query-if` clause asks the receiver whether the *Content* is true or not. A `query-ref` clause asks the receiver to fill the information denoted by '`:wh`' in the *Content*. A `request` clause requests the receiver to do the *Content*. A `commit` clause tells the sender will do the *Content* in future. An `acknowledge` clause tells that the sender receives the *Content* (or a message if no contents are specified). In contrast to it, an `not-understood` clause informs that the sender can not understand the *Content*. We also permit to use `say` speech act for the general purpose, because sometimes it may be difficult to determine speech acts of uttered sentences. A `say` clause can have an optional `:acttype` slot to specify the speech act. For example, `(query-if ...)` is equivalent to `(say :acttype query-if ...)`.

4.2 Syntax for Content

In order to reflect flexibility of natural language, we use an S-expression of frame representation for the content. The syntax of the representation is

```
(FrameName :SlotName1 SlotValue1
           :SlotName2 SlotValue2
           ...)
```

Clause	::= ‘`(`’ *ActType Role_Vale_Pair** ‘`)`’	
ActType	::= ‘`inform`’ | ‘`query-if`’ | ‘`query-ref`’ | ‘`request`’ | ‘`commit`’	
	| ‘`acknowledge`’ | ‘`not-understood`’ | ‘`say`’	
Role_Value_Pair	::= *Role Value*	
Role	::= *Keyword*	
Keyword	::= ‘`:`’*RoleName*	; symbols that begin with ‘:’.
RoleName	::= *Symbol*	
Value	::= *Data*	
Data	::= *AtomData* | *Frame* | *SpecialForm*	
AtomData	::= *Symbol* | *Number*	
Frame	::= ‘`(`’ *Tag Role_Value_Pair** ‘`)`’	
SpecialForm	::= *Collection* | *LogicalForm*	
Collection	::= *Set* | *List*	
Tag	::= *Symbol*	; symbols that begin with ":" are reserved.
Set	::= ‘`(`’ ‘`:set`’ *Data** ‘`)`’	; non-ordered list
List	::= ‘`(`’ ‘`:list`’ *Data*)*	; ordered list
LogicalForm	::= *Negation*	
Negation	::= ‘`(`’ ‘`:not`’ *Data)*	

Table 1. Definition of Civilian's Communication Language

SlotName should be a *Symbol*, and *SlotValue* should be an S-expression of a *Data*, where *Data* is one of *Frame*, *Symbol* or *SpecialForm*. For example, "building (id=300) is in fire" and "agent (id=100) carries agent (id=200) to the building (id=300)" are respectively represented as:

```
(is :subject (building :id 300) :state in-fire)
(do :action carry
    :subject (agent :id 100)
    :object (agent :id 200)
    :destination (building :id 300))
```

SpecialForm includes the following expressions:

```
(:set Data*)
(:list Data*)
(:not Data)
(:wh [Data])
```

`:set` and `:list` forms denote collections of *Data*. `:not` form denotes negation of *Data*. `:wh` is used in `query-ref` clause to specify which information is queried.

Table 1 shows whole definition of the civilian language.

4.3 Examples of Communication

Here, we show several typical examples of communication that civilian agents should be able to understand.

(1) Agent(ID=100) say to agent(ID=200) “person(ID=150) is buried at a building(ID=250)”.

```
    (inform :sender (agent :id 100)
            :receiver (agent :id 200)
            :content (is :subject (agent :id 150)
                             :state (buried)
                             :place (building :id 250)))
```

We also can denote “`:state buried`” instead of “`:state (buried)`”. A symbol is equivalent to a frame its tag is the same symbol and has no slots.

(2) Agent(ID=100) say to agent(ID=200) “person(ID is unknown) is buried at place(100,200)”.

```
    (inform :sender (agent :id 100)
            :receiver (agent :id 200)
            :content (is :subject (agent)
                          :state (buried)
                          :place (position :x 100 :y 200)))
```

We can omit `:id` slot in the agent frame when it is unknown. Generally, any kind of slots can be omitted.

(3) Agent(ID=100) ask to agent(ID=200) “which building is broken, how much, and when?”. Then Agent(ID=200) answer to agent(ID=100) “at time 15, building(ID=150) is broken in degree 100.”

```
    (query-ref :sender (agent :id 100)
               :receiver (agent :id 200)
               :content (is :subject (building :id (:wh))
                             :time (:wh)
                             :state (broken :degree (:wh))))
    (inform :sender (agent :id 200)
            :receiver (agent :id 100)
            :content (is :subject (building :id 150)
                          :time 15
                          :state (broken :degree 50)))
```

(4) Agent(ID=100) say to someone ”help!”.

```
    (request :sender (agent :id 100)
             :receiver (agent :tid T1)
             :content (do :subject (agent :tid T1)
                           :action (help)
                           :object (agent :id 100)))
```

When the same agent (or building) occurs twice in the expression, we can assign a temporal id (`:tid`) to denote the identity in the expression.

(5) Agent(ID=100) say to agent(ID=200) and agent(ID=300) ”move away!”.

```
    (request :sender (agent :id 100)
             :receiver (:set (agent :id 200) (agent :id 300))
             :content (do :action (move)))
```

(6) Agent(ID=100) say to agent(ID=200) ”please say to person(ID=300) to move away”.

```
    (request :sender (agent :id 100)
             :receiver (agent :id 200)
             :content (do: action (request :receiver(agent :id 300)
                                           :content(do:action(move)))))
```

We can embed a clause in any part of content.

5 Discussion for Future Extensions

5.1 Situated Communication

A benefit of using the frame representation is that it is easy to add or omit slots from a frame. The benefit enables to reflect an important feature of natural language conversation, *situated communication.* In human communication, we tend to omit words when they are well known for both of sender and receiver, or when they are not so important. As shown in example 4.3, we can omit slots when they can be filled by receiver. In this example, `:subject` slots in the `do` frames and `:sender` slot in the embedded `request` frame are omitted.

One of open issues in the situated communication is how to treat references like pronouns. We can avoid this by assigning temporal id for referenced information as shown in example 4.3. But this method is not applicable easily in the case the reference points outside of a clause, because we need to define a scope of the temporal id over discourses.

Another issue of situated communication is how to control contexts. Contexts of conversation is an important information source to fill omitted information in utterances, so agents need to manipulate contexts properly. However, the context control is still an open issue on natural language processing. Fortunately, flows of contexts seem to be straight-forward in conversation in the rescue domain. In this case, we can control contexts by introducing `:in-reply-to` slots into the *Clause*.

5.2 Complex Logical Forms

We suppose that all civilian's utterances are simple sentences. Therefore, we introduce only negation (`:not` form) as logical forms. This specification has the following merit:

– It is easy to control the amount of communication transfered by an agent in a time unit. We can treat a simple sentence as a unit of communication, so that we may be able to limit the number of sentences for an agent to say or hear.
– It is easy to define the civilian's behavior as a response to listened messages.

In future, however, we will need to introduce complex or compound sentences, which require more complicated logical forms like '`and`', '`or`', '`if`' and so on.

5.3 Languages for Rescue Agents

While we define a civilian's language to reflect features of natural language, we are thinking that rescue agents will use more formal and well formed language between themselves. Compared with civilians, rescue specialists will be trained to use unambiguous and clear expressions. It is also required that the language can be specify complex combination of conditions and actions, which is similar to a kind of programming language. For this purpose, we can use more formal languages specification like FIPA's Content Language Library [FIP00b].

6 Summary

In this article, we proposed a framework of agents' communication in the RoboCup Rescue simulation, *four layers model of communication*, in which factors of establishment of the communication are classified into knowledge, attention device and transmission layers. Using this framework, researchers can introduce easily new ideas of various level of communication into the simulation.

We also propose a specification of a language for civilian agents. Because civilian agents will be designed to behave as usual people, their utterances will include incomplete and ambiguous expression. The specification is designed to reflect such kinds of natural language features.

Both propositions are prototypes for future extensions. Especially, language specification need to modified to reflect more natural language features.

References

[FIP00a] FIPA, Geneva, Switzerland. *FIPA CCL Content Language Specification*, Aug. 2000. Document number XC00009A (`http://www.fipa.org/`).

[FIP00b] FIPA, Geneva, Switzerland. *FIPA Content Language Library Specification*, Jul. 2000. Document number XC00007A (`http://www.fipa.org/`).

[FIP00c] FIPA, Geneva, Switzerland. *FIPA KIF Content Language Specification*, Aug. 2000. Document number XC00010A (`http://www.fipa.org/`).

[NOD00] Itsuki NODA. Framework of distributed simulation system for multi-agent environment. In *Proc. of The Fourth International Workshop on RoboCup*, pages 12–21, Aug. 2000.

[TTK$^+$00] Tomoichi Takahashi, Ikuo Takeuchi, Tetsuhiko Koto, Satoshi Tadokoro, and Itsuki Noda. Robocup-rescue disaster simulator architecture. In *Proc. of thr fourth International Workshop on RoboCup*, pages 100–105, Aug. 2000.

[WCF$^+$00] Steven Willmott, Monique Calisti, Boi Faltings, Santiago Macho-Gonzalez, Omar Belakhdar, and Marc Torrens. Ccl: Expressions of choice in agent commnication. In *The Fourth International Conference on MultiAgent Systems (ICMAS-2000)*. IEEE, July 2000.