Implementation of NAITO-ADF and its Team Design NAITO-Rescue 2015(Japan)

Kazuo Takayanagi[1], Takuma Kawakami[2], Shunki Takami[1], Yusuke Kitagawa[1], Shivashish Jaishy[1], Nobuhiro Ito[1], Kazunori Iwata[3]
rescue-2015@maslab.aitech.ac.jp

[1] Department of Information Science, Aichi Institute of Technology, Aichi, Japan
[2] Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Aichi, Japan
[3] Department of Business Administration, Aichi University, Aichi, Japan

Abstract. In recent years, the new entrants for the RoboCup Rescue Simulation (RCRS) are seen to have reduced. It can be believed to have occurred because of some existing problems in the Agent Development of RCRS. Hence, we present a design and its implementation in the agent framework as a development environment of the agent, known as Agent Developed Framework (ADF), which would also attract the new entrants to carry out the development of the agent in a development environment(ADK:Agent Developer ' s Kit) that incorporates the ADF.

Keywords: Agent development framework, Common Communication Protocol, K-means algorithm, Hungarian algorithm

1 Introduction

Recently, the new participants for the RCRS have decreased. It is believed that this is a result of some problems that exist in the simulation (RCRS). Below are some of the problems described.

- The simulator provided needs to implement the various algorithms in order to verify them.
- The time consumed for the comparison of algorithms is much longer.

Therefore, the aim of this paper is to solve the abovementioned problems designing and implementing the agent framework as a development environment of the agent, referring it as Agent Developed Framework (ADF). Creating module of the agent. And a questionnaire was conducted to know the opinions about its practical use.

2 Problems of RCRS

To begin with, in RCRS there are three problems at the time of the development of an agent.

- It is a problem to share the source code.
  At present, in RCRS, each team implements its own communication protocol and every agent has own notation and different execution making it incompatible with other teams. As a result, even if the source code of an excellent agent is published on the Internet, it gets hard to be fully utilized in RCRS.

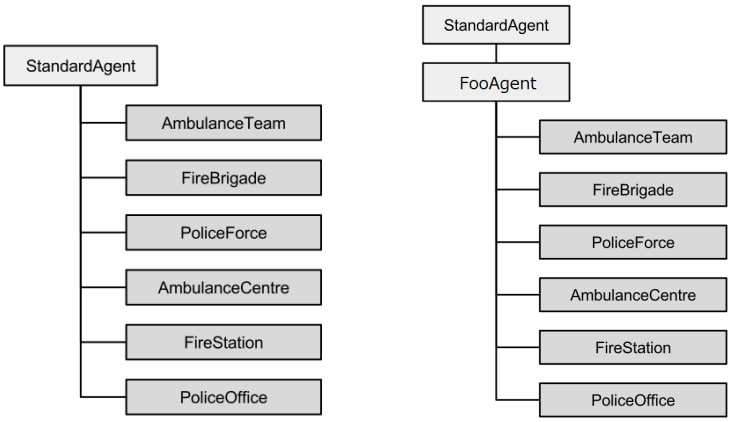
- Verifying the algorithm is a big problem.
  The problems like route search, resource allocation, message sharing and real time planning with the complex issues are yet to be solved in RCRS, which makes it difficult to bring out an individual problem further challenging the comparison of algorithms. Further, various implementations of an algorithm are necessary in order to develop the algorithm with the agent as a base.

3 Design of an agent development support technique.

To resolve the problems, as mentioned in section 2, we would like to introduce the agent framework design as described below.

- Design of the agent with high readability.
  It is known that the redundant parts of the agents have been an issue in the past. StandardAgent is a class of agent, which has a structure using generics[1] to obtain a versatility. The following Fig.1.(a) represents the class of the basic structure of the agent.
  The advantage of this structure is that summarized common method as Fig.1.(b) However, basically shareable part is only code which does not depend on the type of agent, such as route searching and communication method.Furthermore, they can be independently as a class. In addition, due to the impact of generics the PostConnect method performs its own initialization of the Think method that determines performance of the behavior itself. Moreover, it is necessary to describe the method indicated by the agents themselves such as types of agents. Therefore, the specification using generics for new entrants can be considered difficult to understand. Because StandardAgent was repeatedly extends class by the specifications of the RCRS, available variables is not explicit. In contrast, the design of the framework in Fig.2. is composed of two parts.
  In Fig.2., Tactics is a class that describes the behavior algorithm of agent. By using the Tactics, the redundant part of the agent can be separated from the behavioral algorithm, providing the readability. In addition, the route search and resource allocation in Fig.2. shows the interface for implementing each algorithm. This interface prevents the complexity of the source code of Tactics and is also possible to increase the variable readability.

[1] Function for associating particular type to a generic classes and methods.



(a) StandardAgent Configuration was utilized directly

(b) configuration using FooAgent with additional generic methods

Fig. 1. Block diagram of the agent of RCRS class

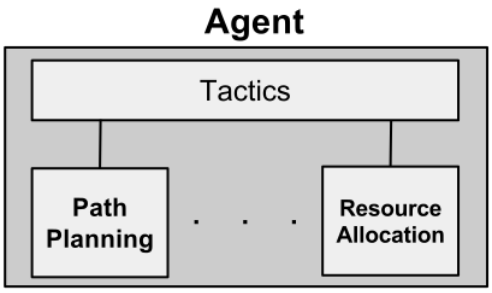


Fig. 2. Agent configuration diagram of the framework

– The unification of specifications.

  In the current RCRS, source code compatibility is low and portability of algorithm is low. In contrast, the interface is used to implement the agent framework in each algorithm in Fig.2. As a result, to solve the problem related to sharing of source code and to have a readability and portability to algorithms is gained. Regarding the communication protocol between agents, previous studies refer to the existing Communication Library. In this study, we actively utilized Comlib, which performs shared communications.

– Design of agent that perform the basic operation.
  It is necessary to develop a verification algorithm for the comparison and verification to a particular problem. It requires enormous time to verify the algorithm. Thus, the basic behavior algorithm to implement the Tactics, to simplify the verification of the algorithm.

4 Implementation of the framework

Hereby, we perform the implementation of the suggestions and models mentioned in Chapter 3.

4.1 Implementation of Tactics

The implementation of Tactics designed in Chapter 3 was performed. The main points that are summarized and shown in Fig. 3. The AbstractComponent or AbstractAgent, in class inherit from StandardAgent. StandardWorldModel holds a variety of disaster spatial information. The variables and methods defined for each class, it becomes explicitly by summarized as in Tactics in Fig. 3. Also, sendMove calls in the think method by combining the methods, such as class Action, keeping the readability of the Tactics.

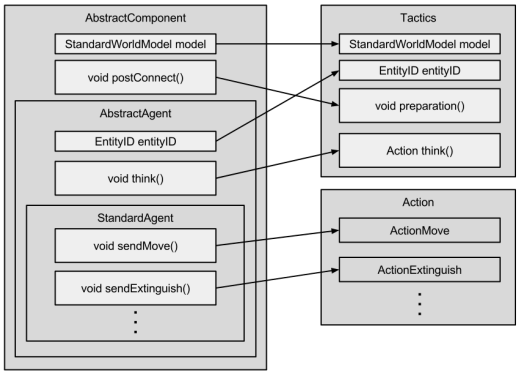


Fig. 3. Framework agent configuration diagram

The Tactics, TacticsAmbulance,TacticsFire and TacticsPolice is implemented. The explanation regarding the interface that implements the algorithm is given in section 4.6.

Also,by bringing together the sendMove method Action class is maintained readability of Tactics.

4.2 Improvement and utilization of ComLib

ComLib, used by unified communications protocol, was applied to the current RCRS. Also attempted to improve the convenience by which the event-driven.

4.3 Implementation of Interface for Implementation of Algorithm

We have implemented about target selection algorithm and path planning algorithms to interface. The target selection algorithm is to determine the target rescue efforts based on resource allocation and damage prediction.

RouteSearcher

The RouteSearcher interface is that which implements the routing algorithm. In the RouteSearcher, algorithms such as Dijkstra[2] and A*[3] are possible to be implemented. In the ADF, implements the A * algorithm as a sample.

TargetSelector

The TargetSelector interface is that which implements the resource allocation algorithm. In the ADF a sample is provided implement the auction algorithm [4].

4.4 Implementation of the Tactics (basic behavior)

As an implementation example of Tactics, we have implemented a BasicTactics.RouteSearcher and TargetSelector has been incorporated into the BasicTactics.Also, for each interface implementation samples are provided, algorithm comparison is easier.

4.5 Package Configuration Framework

In this framework, shown in Fig. 4, the agent developers such as Launcher requires no part or portion required to develop such Tactics. Sample parts can be used as the reference and comparison. We have realized that in the future for the development of the framework, we should keep in mind the manifestation and maintainability of the same.

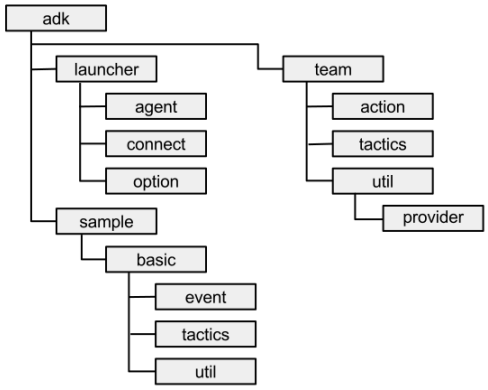


Fig. 4. Framework agent hierarchy diagram

Launcher package

This package contains class Launcher package that loads the agents.

Team package

In the ADF, the team Tactics of each agent is the system put together implementing a Team task. Team task is an abstract class, corresponding to each agent instance of the Tactics to generate a method or team name.

Sample package

In the current sample package the basic packages that include a description of the behavior of basic Tactics are included.

5 TEAM DESIGN

5.1 Abstract of NAITO-Rescue

NAITO-Rescue, described so far, is to implement the agent using the ADF. As already mentioned in ADF the strategy of an agent, Route search algorithm and the resource allocation algorithm are divided into Tactics, RouteSearcher and TargetSelector. As a result, the replacement of the module is facilitated and becomes reusable. Thus, we implement some of the strategies and each algorithm is based on the CodeFinal [5] of Robocup. Currently, we are inspecting the combination of the repeated experiments, optimal algorithm. Following are the description of current implementation of combination and algorithms.

5.2 Common part

RouteSearcher

We used the A$\star$ route search algorithm. When determining the cost of path, following expression is used.

$$RoadArea + sumofallroadarea * Impassable(0or1)$$

$$Impassable = \begin{cases} 0 & passable \\ 1 & impassable \end{cases}$$

Also, we are designing not to select that burning building as a path.

TargetSelector

The resource allocation is necessary to be implemented in TargetSelector. Moreover, there is a necessity to implement them individually because the target is different by the agent. Algorithm of each agent to be implemented will be described in the following section. We have used the cost calculation formula of the RouteSearcher for the derivation of distance cost to be used for the target selection.

5.3 agent strategy

In this section the behavior of each agent is described.

FireBrigade

The task allocation to FireBrigade employs a combination of clustering and allocation method. The clustering method employs k-means algorithm [6], the whereas, the allocation method employs the Hungarian algorithm [7].

- k-means algorithm
  k-means is an algorithm that evenly divides by clustering the elements.
  1. We assign a cluster randomly to each building x_i.
  2. Based on the assigned buildings, we ask the center V_j of each cluster in the calculation.
  3. Obtaining the distance between x_i and V_i,we assign the x_i to the nearest cluster.
  4. We repeat the steps 2 and 3 until there is no change in the center.

– Hungrian algorithm
We define number of tasks as m and the number of agents as n. Hungarian algorithm is used to find a solution by using a matrix of m*n. The cost of the agent is shown in the Fig.5.(a)

1. Subtract the minimum value of the rows from each element in each row and further subtract the minimum value from each column. The results are shown in Fig.5.(b).
2. If you can select 0, one by one, from each row and column, the combination of 0 becomes allocation. Fig. 5(b) proceeds to 3 because it does not meet.
3. All of the 0 are selected in the smallest possible number of columns or rows. In Fig.5.(c) and Fig.5.(d)either one row or two columns can be selected or one column and two rows. In either case, the sum of the columns and rows is three.
4. The minimum values of the elements that are not selected are subtracted from selected elements. Moreover, the elements that are selected in some rows and columns are added to.

	a_1	a_2	a_3	a_4
t_1	5	4	7	6
t_2	6	7	3	2
t_3	8	11	2	5
t4	9	8	6	7

(a) example

	a_1	a_2	a_3	a_4
t_1	0	0	3	2
t_2	3	5	1	0
t_3	5	9	0	3
t4	2	2	0	1

(b) 1st stage

	a_1	a_2	a_3	a_4
t_1	0	0	3	2
t_2	3	5	1	0
t_3	5	9	0	3
t4	2	2	0	1

(c) 2nd stage

	a_1	a_2	a_3	a_4
t_1	0	0	3	2
t_2	3	5	1	0
t_3	5	9	0	3
t4	2	2	0	1

(d) 3rd stage

	a_1	a_2	a_3	a_4
t_1	0	0	5	4
t_2	1	3	1	0
t_3	3	7	0	3
t4	0	0	0	1

(e) 4th stage

Fig. 5. result

5. Return to the step 2. In Fig.5.(e), it can be assigned one for each column and row and it should finish here(e.g. Fig.6.(a),Fig.6.(b)). If not satisfied, the process proceeds to 3 and optimal cost becomes 17.

	a_1	a_2	a_3	a_4
t_1	1			
t_2				1
t_3			1	
t4		1		

(a) resultA

	a_1	a_2	a_3	a_4
t_1	1			
t_2				1
t_3			1	
t4	1			

(b) resultB

Fig. 6. result

PoliceForce

Behavior of PoliceForce (PF) is divided into two stages.

- First: PF uses the route search of the civilian (ShortestPath). Thereby, it shall be removing the obstruction on the path from the initial position to the refuge. (However, PF moves to second when it reaches the path that has finished clearing up the obstruction.).
- Second: PF processes the tasks according to the following priority.
  $Rescuerequest > MainRoad > Other$
  This is for the nearest rescue request to be processed first.
  With regard to the Main Road, the sum of ShortestPath from each Building is calculated and the priority is generated. The collision with each other at the time of clearing up the road is also avoided.

AmbulanceTeam

AmbulanceTeam (AT) selects the rescue target by the following selection procedure.

If the following inequality is satisfied, AT is selected for the rescue. If the inequality is not satisfied, FB or PF is held as the nearest agent available and thus selected.

$$The\ distance\ between\ the\ nearest\ AT * 2 < The\ distance\ between\ the\ nearest\ FB\ or\ PF$$

When the following inequality is satisfied, the nearest agent is selected for the rescue. If the inequality is not satisfied, the nearest Civilian is selected for the rescue.

$$Shortest\ distance\ between\ the\ nearest\ Agent * 2 < The\ distance\ between\ the\ nearest\ Civilian$$

6 Conclusion

In this paper we have intended to show two things. Firstly the implementation of the ADF. Secondly, the development of the team using the ADF.
In the future, we have plans implementing an algorithm described in this paper. If possible we implementing an efficient algorithm than is described in the paper.

References

1. Dai Obashi, Yuki Kawaguchi, Shogo Horibe, Takehumi Ota, Nobuhiro Ito, Toriumi Hujio, "Information shared library for RCRS".
2. E.W.Dijkstra. A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1, pp. 269271, 1959.
3. Raphael B. Hart P.E., Nilsson N.J. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics SSC4, pp. 100107, 1968.
4. T. Sandholm.: "An Algorithm for Optimal Winner Determination in Combinatorial Auctions," In the Proc. of 16th International Joint Conference on Artificial Intelligence (IJCAI '99), pp. 542547, 1999.
5. robocup rescue code final.: `http://sourceforge.net/projects/roborescue/files/2014/team_sources/` (2014)
6. MacQueen, J. B. (1967). "Some Methods for classification and Analysis of Multivariate Observations" Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. pp. 281297
7. Harold W. Kuhn, "The Hungarian Method for the assignment problem", Naval Research Logistics Quarterly, 8397, 1955. Kuhn's original publication.