This document is a preprint of
Takami, S., Takayanagi, K., Jaishy, S., Ito, N., & Iwata, K. (2018). Agent-Development Framework Based on Modular Structure to Research Disaster-Relief Activities. International Journal of Software Innovation (IJSI), 6(4), 1-15. doi:10.4018/IJSI.2018100101

Agent-development framework based on modular structure to research disaster-relief activities

Shunki Takami, University of Tsukuba, Tsukuba, Japan
Kazuo Takayanagi, Aichi Institute of Technology, Toyota, Japan
Shivashish Jaishy, Aichi Institute of Technology, Toyota, Japan
Nobuhiro Ito, Aichi Institute of Technology, Toyota, Japan
Kazunori Iwata, Aichi University, Nagoya, Japan

ABSTRACT

This article describes approaches to the RoboCup Rescue Simulation league which is a part of thevresponse to recent large-scale natural disasters. In particular, the project provides a platform for studying disaster-relief agents and simulations. The aim of the project is to contribute to society by making widely available the findings our research into disaster relief. Some disaster-relief agents contain excellent algorithm modules, which should ideally be shareable among developers. However, this is hindered when the program structure of the agents is different among different teams. Therefore, in this paper, we design and implement a modular agent-development framework that unifies the structure within RoboCup Rescue Simulation agents to facilitate such technical exchange.

Keywords: Development Platform, RoboCup Rescue, Rescue Simulation, Disaster Relief, Multi-agent System

INTRODUCTION

In 2001, the RoboCup Federation started the RoboCup Rescue Simulation (RRS) league to confront large-scale natural disasters (RoboCupRescue, 2017; Kitano 2001). In particular, the annual RRS Agent Simulation and Infrastructure sub-league was established in 2016 for studying disaster-relief agents and simulations. The league's aim is to contribute to the society by submitting results for this project (Visser, 2015).

To solve the disaster-relief problems targeted by the RRS, it is necessary to implement a combination of multiple algorithms, such as those for path planning, information sharing, and resource allocation. The purpose of the Agent Simulation and Infrastructure sub-league is to exchange technical information and to share agent programs. Some of the agents contain excellent algorithms, and so it is desirable to share them among developers (Akin, 2013).

However, because it differs from one team to the next, it is often difficult to re-use program code. This hampers technical exchange.
Thus, in the present paper, we design and implement an agent-development framework (ADF) that unifies the structure within the RRS, thereby allowing program codes to be re-used. This could also act as a platform for researching different disaster-relief algorithms. In the evaluation, we confirm that codes can be re-used.

SUMMARY OF THE RRS

In this section, we explain the simulation content, agent type and behavior, themes of agent development and development environment issues.

Overview of RRS

The RRS is a research platform that simulates disaster-affected areas and disaster-relief activities on a computer (Takahashi, 2001). It can handle disaster-relief activities over approximately 5 hours from the occurrence of a disaster.
Figure 1 shows the activities of agents in the RRS. In the disaster-relief activities, we control six types of agents, namely AmbulanceTeam, FireBrigade, PoliceForce, and the headquarters of each of these units (AmbulanceCentre, FireStation, and PoliceOffice). In addition, there are Civilian agents to simulate civilians in disaster situations.

- AmbulanceTeam
  These agents rescue other agents that cannot move by themselves. They rescue targets from debris and transport them to evacuation centers.
- FireBrigade
  These agents extinguish fires in buildings. They extinguish the fire by discharging the water stored in their tank. It is necessary to supply water when the tank becomes empty.
- PoliceForce
  These agents clear road blockages. They enable other agents to access certain areas by removing obstacles.
- AmbulanceCentre, FireStation, and PoliceOffice
  These agents are headquarters. These agents are only for enabling communication. They cannot move but they are also not injured.
- Civilian
  In the competition, these agents move automatically to evacuation centers.

The RRS simulator consists of a kernel that manages the progress of the simulation. In addition, there are sub-simulator components that simulate disaster situations and agent programs. The simulation proceeds at the rate of one step per minute. The time allowed for calculating one step of an agent is limited to 1 s. Before the start of the simulation, as a pre-calculation stage, it is possible to calculate initial data using only topographical information (Faraji, 2016).
The RRS can be used to research applications of artificial intelligence and information science to natural-disaster rescue problems. In the RRS, five research tasks of are advocated in particular, namely Group Formation, Path Planning, Search, Multi-task Allocation, and Communication (Skinner, 2010). Every year, competitions using agent programs are held for the purpose of technical exchange.

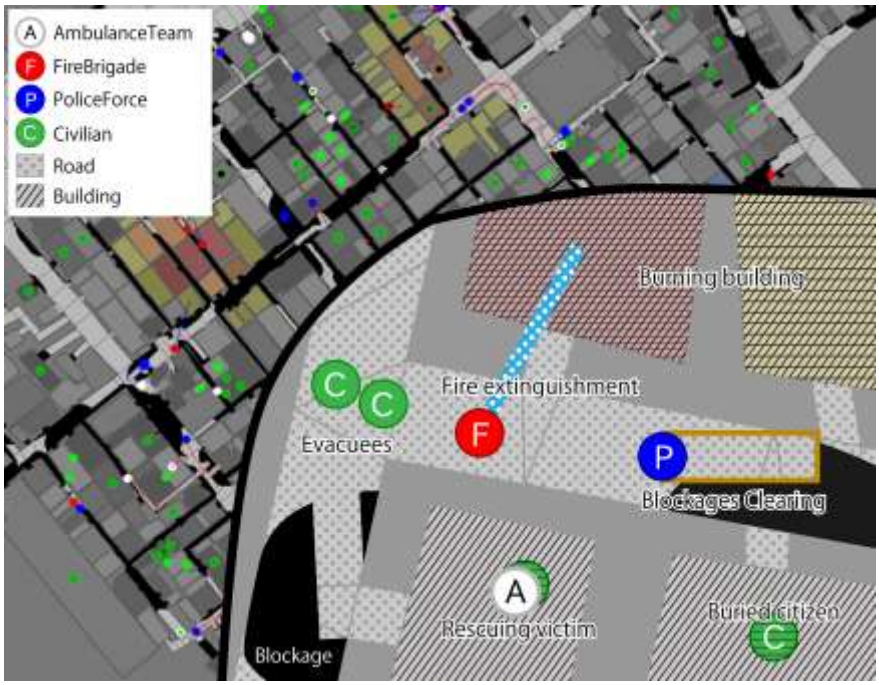

Figure 1. Agents and objects in the RoboCup Rescue Simulation

Agent development in the RRS

The disaster-relief problems handled by the RRS are complex compound problems in which damage situations such as fire, building collapse, and the availability or otherwise of wireless communication services change from moment to moment in afflicted areas. These changes are addressed by the strategies of teams of disaster-relief robots that differ according to the affected area. To construct a disaster-relief strategy, it is necessary to prepare algorithms for all tasks, such as route searching, information sharing, and resource allocation in the disaster environment. Moreover, in activities such as blockages being cleared by the PoliceForce, it is necessary to use angles and coordinates to specify the direction for activity and the positions of the agents.

To promote research involving the RRS, it is necessary to clarify the structure of a complicated disaster-relief problem and subdivide it before solving it. However, it is problematic that the structures of program codes for RRS agent development are not unified. At present, if a program can communicate with the simulator's kernel, it can be created freely. However, for researchers to be able to share their research, it is desirable to unify the structure of the program code and to program each field as a component. Moreover, those components should be re-usable as modules.

RELATED WORKS

OpenRTM-aist is a framework for reusing general robot program code (OpenRTM, 2017). Libraries have previously been proposed to unify the communication protocols between agents in the RRS league.

OpenRTM-aist

OpenRTM-aist is a framework for robot development that was developed by the National Institute of Advanced Industrial Science and Technology. The framework implementation is based on Robotics Technology Middleware (RT-Middleware) (Ando, 2005). Figure 2 shows a conceptual image of the behavior of RT-Middleware. This common platform standard divides

robot elements such as actuator control, sensor input, and algorithms necessary for behavior control into separate components that are known as RT-Components. RT-Middleware then constructs a robot by combining all such RT-Components. This makes it possible to subdivide the elements that are necessary for controlling the robot. Because each component can be exchanged as a module and existing modules can be included, it is possible to reduce the burden on developers when developing and improving robots.
Because RT-Middleware is applied mainly to real robots, it is suitable for developing robots that are controlled in real time. However, it is difficult to utilize existing code and knowledge when adopting RT-Middleware because RRS agents are programed mainly with a sequential structure.

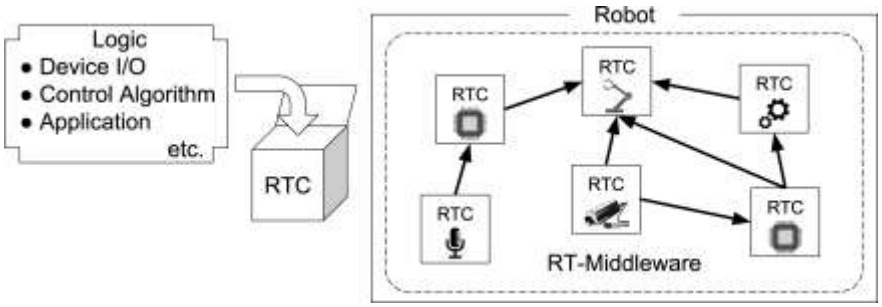

Figure 2. Operation concept image of RT-Middleware and RT-Components (RTC)

Communication library for RRS Agents

Libraries for unifying communication protocols between agents have been proposed (Ohta, 2011; Obashi 2013). The aim is to enable agents developed by different developers to cooperate with each other. All agents that used the library can communicate with each other.
The library has a message class and a manager class. The message class defines the content composition of a concrete message. The manager generates a communication binary from an instance of the message class. Upon reception, each message instance is generated from the binary. There are two series of messages: a series that handles the state of each agent and object, and a series that handles commands for cooperation.
By utilizing this library, developers can concentrate on development of agents only of the type they wish to develop.

APPROACHES

Research objective

In the present paper, we implement an ADF by introducing a modular structure to clarify and solve the complicated problems associated with disaster relief. In this way, we make it possible for researchers to cooperate and solve such problems.

Design of the ADF

To make it easier to reuse the program code and to reduce the burden on researchers, a modular ADF is desired. It is possible to design based on OpenRTM-aist as a framework-design method. However, RRS agents are programmed in a sequential structure. Therefore, we propose and design a unique ADF based on a modular structure that makes it easier to utilize existing program code and knowledge. This framework facilitates a common architecture for agents, modularization of programs, aggregation of information acquisition interfaces necessary for agent decision-making, and a unified inter-agent communication protocol.

Introduction of a common agent architecture

By defining the overall behavior of an agent as a common architecture, we reduce the differences in combinations of components by individual developers and ensure re-usability. This allows developers to implement modules based on this common architecture when developing agent programs.

Figure 3 shows the agent structures before and after introducing this common architecture. The portability of the existing program is low because each researcher builds an agent program independently according to individual research agendas. We have commonized the agent-program structure, which is the shaded part of the figure, so that the program code can be re-used easily. In addition, this unifies the inter-agent communication protocol and enables communication with agents developed by others.

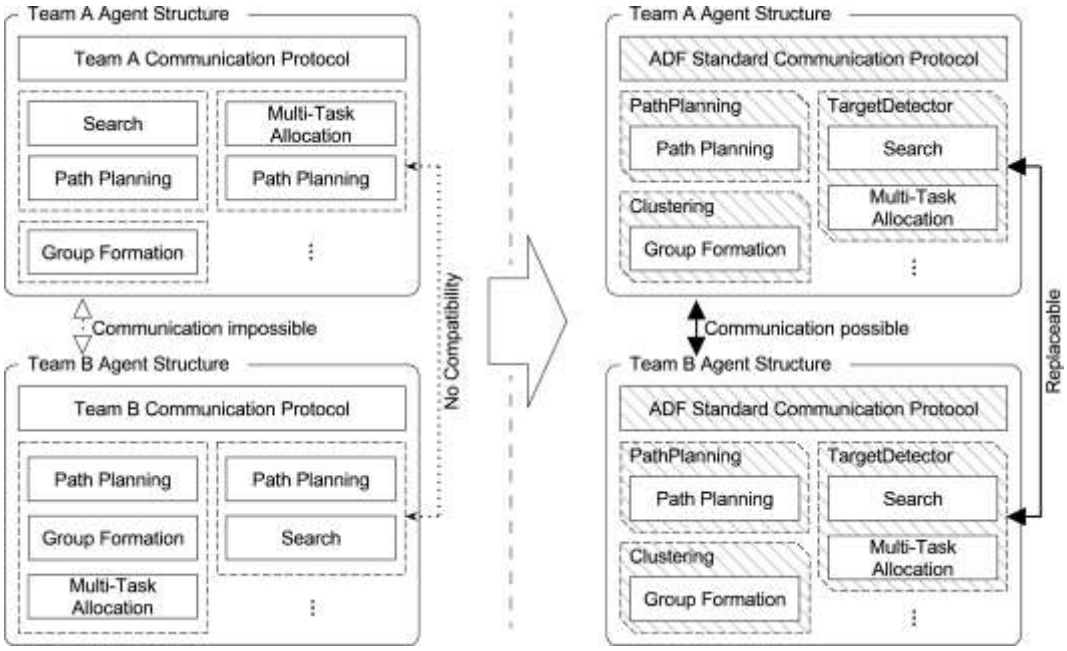

Figure 3. Before (left) and after (right) introducing the common architecture

Modularization of program code

As with the RT-Middleware, component modularization is introduced to reduce the burden on researchers and to make it possible to reuse the program code used in agent development.

- Algorithm modularization

  At the present stage of modularization, we divide as much as possible based on the five tasks presented in the RRS. Figure 4 shows the relationship between RRS tasks and ADF modules. We classify algorithms for solving complex problems and algorithms for solving simple problems as Complex Modules and Algorithm Modules, respectively, thereby clarifying the directionality of each module. We view Complex Modules as aggregates of simple problems. Thus, the modules should be programmed by dividing the structure inside the program code as much as possible. Moreover, if it is subsequently found that the program is divisible into modules, we aim to clarify complicated problems as new modules.

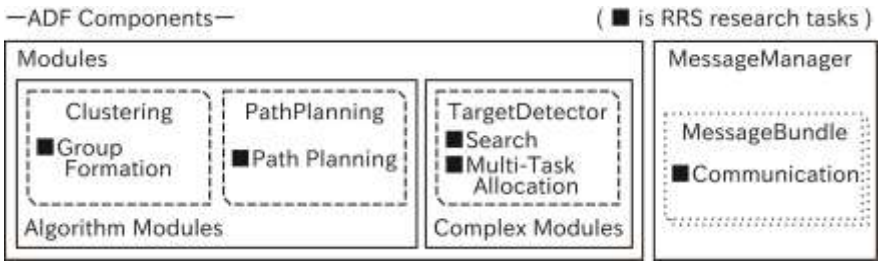

Figure 4. Relationships between the RoboCup Rescue Simulation tasks and agent-development framework components

- Modularization of control program

  Agent control is needed to specify such properties as its coordinates and angles. In this ADF, low-level agent control is modularized as a control program. Macro algorithms (such as decision-making) and micro algorithms (such as control using the coordinates and angles) are separated, thus reducing the burden on researchers who wish to study a single algorithm.

Other approaches

The introduction of a common architecture makes it possible to aggregate information-acquisition interfaces, to manage parameters collectively, and to unify the inter-agent communication protocol.

- Aggregation of information-acquisition interfaces

  The ADF aggregates the interfaces that acquire information necessary for agent decision-making provided by the kernel. This clarifies the acquirable data.
- Collective management of parameters

  The ADF collectively manages the specifications of the modules to be used, the eigenvalues in the algorithm that is to be changed at the time of the experiment, and the pre-computed data. This makes it easier to manage all the various parameters in the experiment.
- Unifying inter-agent communication protocols

  In the RRS, the inter-agent communication protocols are currently not unified. This strengthens dependency between components but makes it difficult to modularize the algorithm. Therefore, introduce unified inter-agent communication protocol. Communication protocol was defined with reference to the previously proposed (Ohta, 2011; Obashi 2013). We define messages communicated under this protocol as members of either an information-sharing family or a command family. In the information-sharing family, information about agents, roads, and buildings is shared. In the command family, commands for relief, fire extinguishing, blockage clearing, and searching are defined. A component for managing the communication is prepared in the same manner as the library in the past. Agents communicate with each other through this component.

IMPLEMENTATION OF THE ADF

This section describes the overall picture and features of an implementation based on the ADF design.

Implementation overview

Figure 5 shows the structure of the ADF that is implemented. The internal aspects of the agent, as represented by the shaded part in the figure, are implementations of the ADF. The modules are objects to be programmed at the time of agent development. The agent developer inherits the base class of the agent provided by the ADF and develops the agent. Methods are invoked according to the whether the agent is defined as an event handler on the base class of the agent. The developer can define the behavior of an agent by describing this method. Since each class instance is automatically generated at agent startup, the agent can access various data by invoking instance methods such as "Information and Data". Included module components are also described in the same style.

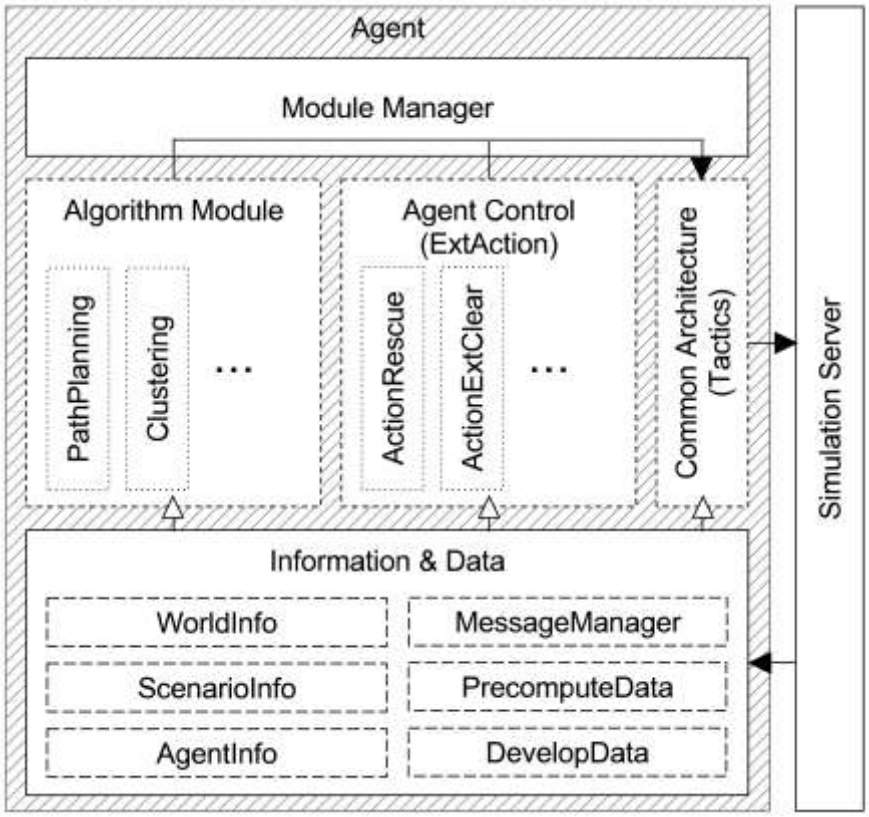

Figure 5. Structure of the proposed agent-development framework

Common agent architecture

This component defines how agents invoke algorithm modules and control-program modules to determine their behavior. This component is referred to as Tactics. Each agent determines its behavior by invoking elements such as task assignment, a route-search algorithm, and a control-program module within Tactics. In addition, each module is invoked by its alias name and can select the module to be attached by the agent-configuration file.

Available actions are expressed by the following class instances, and they are determined by returning it to the agent thinking base routine.

- ActionRest
  Stay in place
- ActionMove
  Move along the path (Arg.: Path, [X], [Y])
- ActionRescue (for AmbulanceTeam)
  Rescue the victim (Arg.: TargetVictim)
- ActionLoad (for AmbulanceTeam)
  Accommodate the victim (Arg.: TargetVictim)
- ActionUnload (for AmbulanceTeam)
  Set down the victim
- ActionExtinguish (for FireBrigade)
  Extinguish fire (Arg.: TargetBuilding)
- ActionRefill (for FireBrigade)
  Refill tank with water
- ActionClear (for PoliceForce)
  Clear blockage on road (Arg.: X, Y)

Modules

As shown in "Modularization of program code", each task is modularized as Clustering, PathPlanning, or TargetDetector. In addition, the control program is modularized as ExtAction. An instance of the module is created and managed in a component called the ModuleManager. This further enables switching the module to be used by loading the module-configuration file (module.cfg) at agent startup.

Collective management of parameters

Figure 6 shows the flow of loading module instances. The settings of the module to be loaded are obtained from the aforementioned file module.cfg. Parameters in algorithms can be obtained from the DevelopData component, and each value can be input by JSON (The JSON, 2013) formatted text as an argument at agent startup. In addition, the pre-computed data can be stored in the PrecomputeData class.

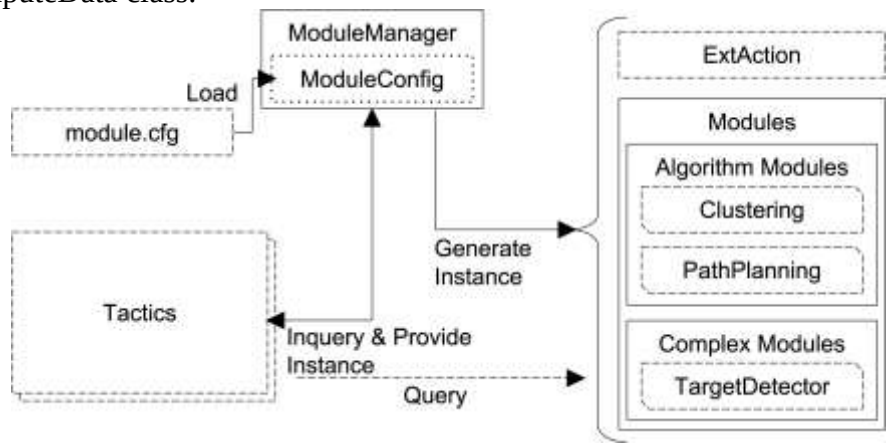

Figure 6. Flow of loading module instances on the agent-development framework

Inter-agent communication protocols

We defined messages as members of either an information-sharing family or a command family. The following messages are implemented in the information-sharing family:

- MessageRoad
  Contains the state of a road and includes information about any blockages.
- MessageBuilding
  Contains the state of a building.
- MessageCivilian
  Contains the state of a civilian.
- MessageAmbulanceTeam
  Contains the state of the AmbulanceTeam and includes the actions of the agents.
- MessageFireBrigade
  Contains the state of the FireBrigade and includes the actions of the agents.
- MessagePoliceForce
  Contains the state of the PoliceForce and includes the actions of the agents.

The following messages are implemented in the command family. The distinction between command and request depends on whether there is a broadcast designation.

- CommandAmbulance
  Sends commands and requests to the AmbulanceTeam.
- CommandFire
  Sends commands and requests to the FireBrigade.
- CommandPolice
  Sends commands and requests to the PoliceForce.
- CommandScout
  Sends scout commands and requests.
- MessageReport
  Reports the results of the commanded actions.

For communication, as shown in "Implementation overview" we use MessageManager in "Information and Data." Messages to be sent are registered with MessageManager for transmission. Received messages are retrieved from MessageManager. Specifying the class of a message allows us to retrieve only specific messages if we so wish.

EXAMPLE OF AGENT IMPLEMENTATION USING ADF

In this section, we describe an example agent that implements a unique algorithm for the PoliceForce route-searching algorithm. Every other module can be implemented by following an equivalent procedure.

TacticsPoliceForce

The TacticsPoliceForce module uses the common architecture and implements the module-invoking structure shown in Figure 7. The shaded area is the part that processes the command from headquarters. The TacticsPoliceForce agents always act in a de-centralized way because this shaded area returns a null result in this implementation. Other modules exhibit autonomous behavior. The TacticsPoliceForce agent is currently a common architecture for PoliceForce in the whole project.

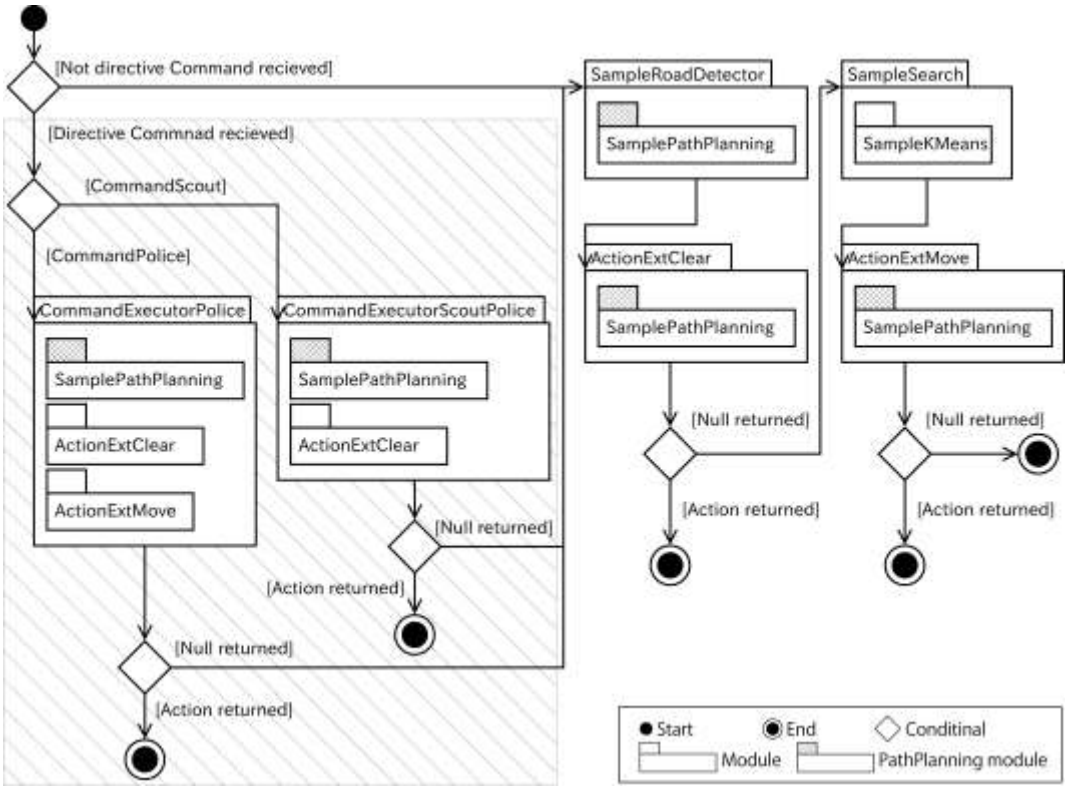

Figure 7. Module-invoking structure of the TacticsPoliceForce sample agent

PathPlanning

Developers implement a unique path-planning algorithm, details of which are not described here because it is different from the essence of this paper. This section explains how to implement the program.

As shown in "Implementation overview", the inputs of the module are the starting location, ending location, and the components shown in "Information and Data". Implementation programs calculate routes based on the input into the calc() method. We implement the getResult() method so that the route path can be acquired.

Loading modules

Module incorporation changes the specification of PathPlanning in the file module.cfg. The specifications of the module that undergoes changes are as follows:

- CommandExecutorPolice.PathPlanning
- CommandExecutorScoutPolice.PathPlanning
- ActionExtClear.PathPlanning
- ActionExtMove.PathPlanning
- (SampleRoadDetector.PathPlanning)

EVALUATION

We developed and tested the agent program for an actual convention to evaluate the modularity of agents and algorithm components. The experiment was done in two ways.

The first experiment confirmed the modularity of the agent. We developed and tested the agent program at an actual workshop to evaluate the re-usability of the code. We collaborated with multiple developers with different programming philosophies to assess whether the agents could be developed in the proposed environment.
The second experiment confirmed the modularity of the algorithm component. It confirmed whether the algorithm component of an agent developed by one developer can be replaced with the algorithm component from other developers.

Experiment on agents' modularity

Experimental method: At the workshop of the RoboCup Simulation league held at Fukuoka University, Fukuoka, Japan on October 22–23, 2016, approximately 30 people were divided into six teams to develop agents. We conducted an experiment to collaborate and work in combination with AmbulanceTeam, FireBrigade, and PoliceForce agents that were developed by each team. In doing so, we confirmed that it was possible to combine agents developed by different developers. At the workshop, we experimented with only six combinations because of time restrictions. However, we have since experimented with 216 combinations.

Results and discussion: Table 1 gives the eight highest combined teams' scores of the combined experiments of each team (A–F), scores of a sample team, and the scores for each team as a reference. The score calculation method follows the specification of the RRS simulator (Parker, 2014). The score of each team is no better than the combination of the eight highest scores because the time for development was short. However, because the scores are high when these agents are combined, it can be seen that agents developed by different developers are cooperative. We confirmed that the agents developed using the proposed ADF worked in conjunction with each other. Developers were able to develop agents in a limited time because of cooperation within each team. This confirms that the agents are re-usable.

Table 1. Eight highest scores of agent combination experiments

Rank	**Ambulance**	**Fire**	**Police**	**Score**
1	A	C	A	143.4091
2	A	B	A	141.4178
3	E	B	A	140.9280
4	A	B	F	136.5078
5	E	B	B	134.8257
6	B	B	D	134.3782
7	C	B	D	134.3404
8	B	B	E	133.8126
reference values	Sample	Sample	Sample	114.2580
	A	A	A	114.2574
	B	B	B	126.5861
	C	C	C	124.3933
	D	D	D	114.2580
	E	E	E	119.9167
	F	F	F	120.8222

Experiment on component modularity

Experimental method: Twelve teams from each country participated in the RoboCup Rescue Simulation league held in Nagoya, Japan on July 27–30, 2017 (Results 2017, 2017). We conducted experiments in which TargetDetector of each participating team was incorporated into the base agent and operated. The base agent is an agent implemented by the team that is used in the ordinary competition. We replaced TargetDetector in it and generated an agent. The base agent designated the top five teams (MRL, AUR, RAK, SEU, and CSU) of the ordinary competition because of time restrictions. In doing so, we confirmed that it was possible to combine agents developed by different developers. However, because it was conducted as a competition, the disaster scenario (1–5) differs randomly for each base agent. We confirmed that algorithm components developed using the proposed ADF worked in conjunction with each other.

Results and discussion: Table 2 is a summary of only the team selected as the base team among the results of the combination experiment. It is in the order of the result of the ordinary competition from the top left of the table. Point notation for the competition is used for the result because the scenario is different. The point calculation method follows the official rule of the RRS (Faraji, 2017). MRL and AUR have high scores when combined with some other teams and when not in combination. RAK, SEU and CSU show high scores when using other base agents. This indicates that the algorithm components of other teams supplement the insufficient part of the agent algorithm. From this, it is expected that research will be possible based on algorithm components in the future.

Table 2. Excerpt of only the team selected as the base team among the results of component combination experiment

Team	**MRL base (Scenario 1)**	**AUR base (Scenario 2)**	**RAK base (Scenario 3)**	**SEU base (Scenario 4)**	**CSU base (Scenario 5)**
MRL	24	1	18	11	24
AUR	15	24	24	22	4
RAK	11	15	17	24	8
SEU	14	13	3	14	24
CSU	6	20	6	7	11

CONCLUSION

In this paper, we designed and implemented an ADF that unifies the structure within the RRS league to foster technical exchange. We confirmed that agents developed by different groups of developers could collaborate, and that algorithm modules developed by different groups of developers could be re-used in by other developers in their agents. Furthermore, ADF has been adopted as a standard development environment of the RRS league since 2017. As mentioned in the ADF implementation, the algorithms for solving complex problems (known as the Complex Modules) are aggregated from simpler problems. Thus, in the future, if it is found empirically that a particular program can be divided, we aim to clarify complicated problems as new

modules. This will lead to better resolution of disaster-relief problems. Eventually, we aim to return the research results of the RRS league to society by clarifying disaster-relief problems and proposing individual algorithms that are applicable to disaster relief.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number JP16K00310 and JP17K00317. We thank Edanz Group (www.edanzediting.com/ac) for editing a draft of this manuscript.

REFERENCES

Akin, H. L., Ito, N., Jacoff, A., Kleiner, A., Pellenz, J., & Visser, A. (2013). Robocup rescue robot and simulation leagues. Vol. 38. AI Magazine (pp. 78-86). Retrieved September 2, 2017, from http://www.aaai.org/ojs/index.php/aimagazine/article/view/2458

Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., & Yoon, W. (2005, Aug). RT-Middleware: Distributed Component Middleware for RT (robot technology). In 2005 ieee/rsj international conference on intelligent robots and systems (pp. 3933-3938). doi: 10.1109/IROS.2005.1545521

Faraji, F., Nardin, L. G., Modaresi, A., Helal, D., Iwata, K., & Ito, N. (2017). Robocup Rescue Simulation League Agent 2017 Competition Rules and Setup. Retrieved September 2, 2017, from http://roborescue.sourceforge.net/web/2017/downloads/rules2017.pdf

Parker, J., Godoy, J., Groves, W., & Gini, M. (2014). Issues with Methods for Scoring Competitors in RoboCup Rescue. In ARMS workshop at AAMAS.

Kitano, H., & Tadokoro, S. (2001). Robocup rescue: A grand challenge for multiagent and intelligent systems. AI magazine, 22(1), 39.

Obashi, D., Hayashi, T., Iwata, K., & Ito, N. (2013). An implementation of communication library among heterogeneous agents naito-rescue 2013(japan). In Robocup 2013 Eindhoven.

Ohta, T., & Toriumi, F. (2011). Robocuprescue2011 rescue simulation league team description. In Robocup 2011 Istanbul.

OpenRTM-aist. (2017). Retrieved September 2, 2017, from http://www.openrtm.org/

Results 2017 RoboCup Rescue Agent Simulation League Tournament. (2017). Retrieved September 2, 2017, from http://roborescue.sourceforge.net/web/2017/results.html

RoboCup Rescue Simulation. (2017). Retrieved September 2, 2017, from http://roborescue.sourceforge.net/

Skinner, C., & Ramchurn, S. (2010). The robocup rescue simulation platform. In Proceedings of the 9th international conference on autonomous agents and multiagent systems: Volume 1 - volume 1 (pp. 1647–1648). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. Retrieved September 2, 2017, from http://dl.acm.org/citation.cfm?id=1838206.1838523

Takahashi, T., Takeuchi, I., Koto, T., Tadokoro, S., & Noda, I. (2001). Robocup rescue disaster simulator architecture. In Robocup 2000: Robot soccer world cup iv (pp. 379–384). London, UK, UK: Springer-Verlag. Retrieved September 2, 2017, from http://dl.acm.org/citation.cfm?id=646585.698826

The JSON data interchange format (1st Edition ed.; Tech. Rep. No. Standard ECMA-404 1st Edition / October 2013). (2013, October). ECMA. Retrieved September 2, 2017, from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Visser, A., Ito, N., & Kleiner, A. (2015). Robocup rescue simulation innovation strategy. In R. A. C. Bianchi, H. L. Akin, S. Ramamoorthy, & K. Sugiura (Eds.), Robocup 2014: Robot world cup xviii (pp. 661–672). Cham: Springer International Publishing. Retrieved September 2, 2017, from http://dx.doi.org/10.1007/978-3-319-18615-354 doi: 10.1007/978-3-319-18615-354