Multiagent-based Autonomic and Resilient Service Provisioning Architecture for the Internet of Things

Takumi Kato[†, ††], Hideyuki Takahashi[†, ††], Tetsuo Kinoshita[†, ††]

[†]Graduate School of Information Sciences (GSIS), Tohoku University, Sendai, Japan
[††]Research Institute of Electrical Communication (RIEC), Tohoku University, Sendai, Japan

Summary
This paper proposes the Agent-based Internet of Things (AIoT) architecture and AIoT middleware to realize autonomic and resilient service provisioning in IoT systems. As a massive number of devices are becoming part of the Internet, it is highly difficult to organize IoT devices as a system to provide appropriate services in our lives. It is difficult because the IoT devices are heterogeneous, and the situation often changes overtime which requires complicated reconfigurations of IoT systems. Our AIoT middleware provides IoT devices with the functionality of autonomic and resilient service provisioning in IoT systems. In this paper, we propose AIoT architecture of IoT application, AIoT organization and re-organization scheme to compose and operate IoT application according to user requirement and environmental condition. As the evaluation, we have implemented an autonomous logistics application with simplified logistics robots using proposed AIoT middleware, and conducted logistics experiments and scalability measurement simulation, to evaluate the effect of introducing AIoT architecture and AIoT middleware. We have also performed an architectural flexibility analysis on our architecture to examine further feasibility of proposals. The results of evaluations show the evidence of the realized capability of autonomic and resilient service provisioning, as well as the architectural flexibility of our middleware.

Key words:
Multiagent System, Agent-based Internet of Things (AIoT), Distributed System, Architecture, Autonomic System, Autonomous System Construction, Reorganization.

1. Introduction

Our daily lives involve a number of various computational devices connected to the Internet, e.g. smart phones, personal computers, home information appliances, building components, and even the system components of power grid. As such devices gain the functionality to connect to the Internet based on their controlling software, there are diverse services that attempts to utilize them to benefit our lives. This paradigm is called the “Internet of Things (IoT),” in which heterogeneous devices are utilized by series of software through the networks. The challenge that the number of IoT related studies share, is the effective utilization of several and diverse computational entities that operate in a physical space to realize diverse services for people.

As the number of IoT devices and its services grows, it becomes a tremendous burden for users and developers to organize individual IoT devices based on the requirements of services. In order to mitigate the burden on realizing and operating IoT system, there are number of investigations on automatic, autonomous and flexible operation of IoT devices [1][2]. The area of the investigations is diverse, e.g. home automation, health care, building security services, transportation, market administration, energy distribution, logistics, manufacturing, and so on.

In the common sense among the prior studies, IoT device is a networked computational device which has sensing and/or actuation function(s) in physical space. IoT system is a particular set of IoT devices which control each device’s action to provide services to fulfill the requirement of people, in which service is a series of actions taken by the IoT devices that collectively realize particular output to people, such as provisioning of desired data (e.g. queried data from sensor network, collected data from participatory sensing), and real-world operation by physical actuation (e.g. appliance control in home automation, manufacturing actions in factory automation).

Although there are investigations on smart functionality of IoT devices, it is still managed manually to compose IoT systems by selecting and configuring series of IoT devices based on the demand of people who introduces the desired IoT system. As the diversity of IoT devices and the provisioning services grow, the burden on realizing IoT system increases exponentially. Therefore, we focus on the autonomic composition of IoT system based on user requirements to reduce the burden of humans on realizing IoT systems.

In addition, since IoT system operates in physical environment, adjustment and reconfiguration of the system are often required, which also complicate the IoT system operation. It is even greater burden for users to re-configure the IoT systems, which is also performed

Manuscript received June 5, 2017
Manuscript revised June 20, 2017

manually in the existing investigations. Therefore it is expected for IoT system to be resilient to sustain the services against various changes in environment, system and user situation.

To address the problem, we focus on autonomic and resilient service provisioning in IoT systems, and propose the Agent-based Internet of Things (AIoT) architecture that realizes autonomic organization of IoT systems, as well as resilient operation of the organized IoT system by taking advantage of agents. We also propose the AIoT middleware that provides common runtime platform and agent template for IoT systems to organize and operate in such manner.

By realizing the autonomic and resilient service provisioning, IoT system becomes able to autonomously compose IoT application to provide desired services, (e.g., combining multiple robots to perform housework in ambient assisted living environment [3]). IoT system also becomes able to resiliently replace and tune the IoT devices, e.g. replacing the housework robot in case of failure, or to change the parameter to perform task faster, etc.

The rest of this paper is organized as follows: Section 2 describes the related work and the problems on composing and operating services in IoT systems. Section 3 explains the proposed AIoT architecture and AIoT middleware to resolve the problems. Section 4 shows the implementation and experiments to show the realized autonomic and resilient service provisioning capability, as well as the architectural analysis on the proposed architecture. We conclude this paper in Section 5.

2. Related Work

In order to examine the underlying problem in the attempt of realizing and operating IoT systems for service provisioning, this section looks into the existing works, to derive the remaining and underlying problem of the existing works on service provisioning.

2.1 Service Provisioning based on Automatic Mash-up and Enabling Architecture of IoT devices

To utilize heterogeneous IoT devices together, Device Ensemble System [4] investigates a method to enable sensors and smart home appliances to work together by connecting them into a centralized webserver, and control these devices based on the requests sent from a smartphone of a user. There is also a study on organizing IoT devices registered in REST based web servers, called the Web of Things (WoT) [5][6][7]. WoT approach offers an abstraction of IoT device and enables heterogeneous IoT devices to be utilized by web services. The abstraction makes it easier for a user to manually construct a service by interconnecting the abstracted functions of IoT devices.

2.2 Agent-based Approach for Autonomous Service Provisioning in IoT systems

The conventional approaches require constant re-configurations whenever a system faces change of user requirement, partial failure, change of situation, etc. Therefore the prior survey [2][8] points out that the characteristics of intelligent agents are well suited for the utilization of IoT devices. In order to sustain the services, there are investigations on introducing agent technologies to develop IoT devices into Smart Objects (SO) [9]. Smart object provides context-awareness, and adaptability to perform and adjust device's action as an individual, using the given knowledge to the agent. By introducing the agent technology, and knowledge on the context of user and situated environment, SO provides a capability to sustain service by adjusting its actions based on contents.

As the application of SO, there is a centralized approach for device discovery and orchestration in home and building automation [10][11]. They utilize a single knowledge base shared among SOs. The cooperation among devices is organized manually by a user through the web page of user interface. The agents in the system monitor the state of each appliance, and execute actions based on the user requirements.

Agent-based decentralized approach of IoT devices is proposed as various middleware in the existing works. UBIWARE [12], UBIROAD [13] and S-APL [14] provide semantic knowledge representation and reasoning capability to the utilize functions of IoT devices in situated environment. Impala [15], Smart Messages [16], ActorNet [17], Agilla [18] are introducing intelligent and mobile agents to manage sensor network to sustain sensing service by adapting the system towards the partial failure or the status change of the system component. These investigations of middleware demonstrate the IoT device's capability to adapt toward other component's failure, and situational changes in the IoT systems. Although these studies on middleware only deals with particular type of IoT application, i.e. sensor networks, not including various actuators.

2.3 Underlying Problem and Our Approach

The existing studies mentioned above are:

- Making individual IoT devices unified component for users to manually and easily select and configure IoT system

- Making individual IoT devices smarter by introducing intelligent agents to sustain service provisioning as an individual device or as a particular application with homogeneous application

These two points above are separately solved, because it is difficult for users to manually compose IoT system if the IoT devices are autonomous and dynamical. On the other hand, it is difficult to compose IoT system using autonomous IoT devices because of the heterogeneity and dynamical nature of the introduced agents.

Therefore, despite of the number of investigations explained above, it is still a remaining problem to effectively organize the functions of IoT devices to compose service as IoT system, as well as to re-configure the organization of heterogeneous IoT devices towards partial failure and situational change. Therefore, the composition of IoT system is performed manually.The remaining problem of the existing works can be set as follows:

- **Problem 1**: Autonomic organization of IoT system by taking into account the user's requirement and diverse IoT device functionalities
- **Problem 2**: Resilient tuning and replacement of system components to deal with users, systematical, and environmental change

In order to deal with these problems mentioned above, we propose AIoT architecture and AIoT middleware to systematically help users to build IoT system based on the proposals.

3. Autonomic and Resilient Service Provisioning based on Agent-based Internet of Things Architecture

We propose Agent-based IoT (AIoT) architecture, and AIoT middleware to realize autonomic and resilient service provisioning in IoT systems. Following subsections describe the details of proposal.

3.1 Requirements for solving problems in existing works

In the paradigm of IoT, there are pervasive and heterogeneous devices. It is difficult (time consuming, requiring highly cognitive processing) to monitor the condition and function of each IoT device, as well as to select and configure the devices' functions to compose the desired IoT application. Since it is necessary to consider the operational status of each IoT device. It is even more complicated to re-configure, replace, and upgrade these IoT devices to deal with the change in the user requirement and system's operational environment.

Based on the problem and these characteristics of IoT devices, we have derived the requirement of the IoT system to address the problems mentioned previously. Following paragraphs are the explanation of the functionality requirements.

- **Requirement 1.** *Transparent control of IoT devices*: Transparent control is crucial for the IoT system in service provisioning, because the IoT devices are heterogeneous, and it is difficult to monitor the heterogeneous devices in IoT systems. Generally, the APIs, functionality, physical constraints, and the setting procedures of the IoT devices vary from one another. Given the current diversity of IoT devices (e.g. static small sensors, home automation actuators, autonomous robots, etc.), autonomic organization of IoT systems cannot be realized without transparent control scheme through homogeneous interfaces to IoT devices.
- **Requirement 2.** *Autonomous selection and control of IoT devices according to diverse user requirements*: Diversity of user requirement is one of the troubling characteristics of IoT system service provisioning, because IoT devices provide multiple functions, and the sharing devices for multiple purpose generally result in conflicting controls. User requirement is a set of IoT system services that users desire from the system. Since the diversity of IoT device is high, the requirement also varies, e.g. "lower the temperature of this room," "I want to carry this package there," "I want to assemble this furniture," etc. In the environment where IoT devices are pervasive, it is impractical for IoT system to manually configure each function for each requirement from different users. Given the characteristics of IoT systems, in order to realize the autonomic organization of IoT systems, it is necessary that individual IoT devices are capable to select, and execute the actions according to the user requirement and their operational conditions. The user requirement needs to be satisfiable by the set of available services.
- **Requirement 3.** *Sustainable service provisioning against changes*: Since IoT devices are running in physical space, it is exposed to various changes: environmental condition change, device condition change, as well as user situational change. In order to sustain the service provisioning, IoT system often requires re-configuration. Given the complexity of

IoT system, it is required to resiliently tune and re-configure the IoT system in autonomous manner. For example, replacing networked information home appliance (e.g. smart lights, thermostat sensor, etc.) in case of breakdown. Other than replacement, tuning of IoT device performance is also a re-configuration, such as adjusting air conditioner's temperature setting in response to power consumption saving demand.

In the next subsection, we propose the detail of AIoT approach to fulfill the requirement to solve the underlying problem in the IoT paradigm.

3.2 AIoT Architecture: Multiagent-based Autonomic and Resilient Service Provisioning Architecture for the Internet of Things

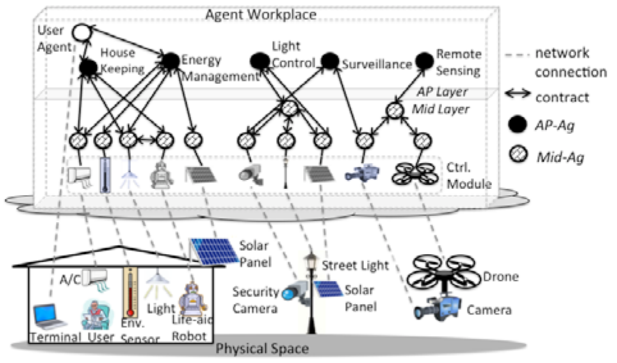
i) Essence of Proposed Solutions

In order to satisfy the requirement 1, 2, and 3, we have proposed 3 solutions as follows:

- **S1.** *AIoT architecture of application and Agentification of IoT device*: In order to realize transparent control of IoT devices, **Req.1**, we propose AIoT architecture of IoT applications, which is composed of AIoT devices. The AIoT device is realized through the design and implementation process called *agentification*. With the commonly agentified IoT device, heterogeneous IoT devices can be controlled in homogeneous manner. We introduce the AIoT device, and the manager agent of these devices which realize the desired application by users. Individual AIoT devices provide problem solving capabilities.

- **S2.** *AIoT Organization Scheme for Autonomic AIoT Application Composition According to User Requirement*: Since IoT devices are controlled by intelligent agents by **S1**, it is able to embed knowledge to autonomously select and execute IoT device. By taking advantage of AIoT devices' capability, which are controlled by intelligent agents, we propose a novel concept of AIoT organization to realize IoT-oriented device organization. We propose AIoT agent organization scheme to compose AIoT application, by systematic interactions among agents to select and configure AIoT devices, to fulfill **Req. 2**.

- **S3**. *AIoT Re-organization scheme for Resilient AIoT Application Operation*: In order to resiliently operate



- Fig. 1 Overview of AIoT architecture of IoT applications.

- AIoT application against various changes to fulfill **Req. 3**, we propose AIoT re-organization scheme to tune and replace the AIoT devices as the extension of **S2**. The instance of re-organization is replacement, re-configuration, addition and removal of devices, etc.

Next sub-subsection explains the overview of **S1, S2** and **S3**.

ii) Overview of Proposed AIoT Architecture

We envision the agent-based approach to resolve the existing works' problems. Given the characteristics of IoT systems, intelligent agent equips the desirable characteristics to deal with the existing difficulties, such as autonomy, flexibility, and sociality. By taking advantage of intelligent agents, we envision the AIoT architecture of IoT applications, which is realized by series of agents that manage and operate IoT devices based on user requirement, and its environment. First, we explain the overall view of AIoT architecture and its components, and then describe the autonomic AIoT application composition and resilient operation scheme.

Fig. 1 illustrates the overview of AIoT architecture and applications. The agents that control IoT devices are Middleware Agents (Mid-Ag), and the device controlled by Mid-Ag is called AIoT device. The AIoT devices are organized by Application agents (AP-Ag), which have task descriptions to realize services. These agents compose AIoT application. There are 2 layers of agents, namely, application (AP) layer, and middleware (Mid) layer.

AP layer provides the services as AIoT applications to the users, via AP-Ags in the layer. AP-Ags provides service parts of applications which realize the applications. AP-Ag works alone, or with the other AP-Ags and/or Mid-Ags to provide services. In case of working with Mid-Ags, an AP-Ag works as a manager to control, organize, and re-

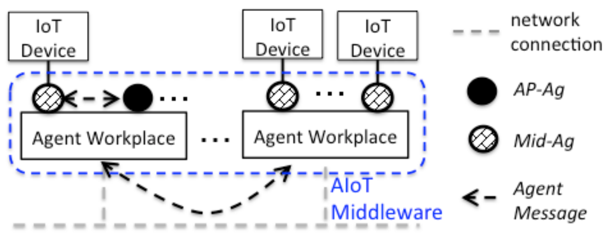


Fig. 2 Overview of AIoT middleware hosting Mid-Ag and AP-Ag to compose AIoT architecture of IoT systems.

organize the Mid-Ags. Service requirement is acquired by user agents, and the user agents select AP-Ags which satisfies the requirement, to start AIoT application construction and service provisioning.

To realize desired AIoT applications, Mid layer inter-connects the IoT devices (e.g. various sensors and robots) and the AP layer's agents. Mid-Ag is realized by the agentification process to introduce intelligent agent functionality to target IoT device. Mid-Ag works alone, or with the other Mid-Ags as an organization to control and monitor the IoT devices.

In order to control IoT devices in response to user requirement and operational condition, AP-Ag controls Mid-Ags as a manager, or Mid-Ag controls an IoT device based on its sensing activity. Both Mid-Ag and AP-Ag are called AIoT agent. The architecture of AIoT agent, as well as the agentification process to acquire Mid-Ag is explained in Section 3.3.

In the AIoT architecture, building organization of agents is a process to select and configure AIoT agents to solve problems in distributed manner, based on the environmental condition and task description reflecting user requirement. The agent organization process is performed through AIoT organization scheme, which is explained in Section 3.4. Re-organization of agent is a process to re-configure and/or replace AIoT agents in response to the changes in various requirements. The re-organization is realized by AIoT re-organization scheme, which is explained in Section 3.5.

Both of AIoT organization scheme, and AIoT re-organization scheme are conducted based on the AP-Ag and Mid-Ag's knowledge on cooperation, which is commonly embedded among AIoT agents by using the agent templates provided by the AIoT middleware AIoT middleware.

The architecture of AIoT middleware is shown in Fig. 2. Since the AIoT architecture offers services by Mid-Ag and AP-Ag, AIoT middleware offers agent workplace that hosts AIoT agents and mediate messages among the agents. Agent workplace also provides functionality to interconnect workplaces through TCP/IP network, which provides agents with messaging capability through networks. AIoT middleware provides the template of agent and IoT device control module, common rule sets and vocabulary to realize AIoT agents, as described later in section 3.3.

AIoT application is composed by AP-Ag using its AIoT organization scheme to organize AIoT agents placed by developers or users, and operated by the AP-Ag based on its knowledge including AIoT re-organization scheme. When the task is finished, agent organization is dissolved and the member agents' operations are terminated. Following subsections explains the following contents:

- AIoT agent architecture and agentification process in Section 3.3
- AIoT organization scheme in Section 3.4
- AIoT re-organization scheme in Section 3.5

3.3 AIoT Agent Architecture and Agentification Process

IoT devices gain the functionality to execute task and cooperate with other devices through the design and implementation process called *agentification*. This subsection describes the proposed AIoT agent's architecture, as well as the process of agentification to realize the AIoT agent.

i) Architecture of AIoT agent

In order to develop heterogeneous IoT devices into homogeneous entities, as well as to introduce autonomous functionality into IoT device control, we propose the architecture and development process of AIoT agent. Fig. 3 shows the architecture of software agent which controls an IoT device. AIoT agent is combination of knowledge-based agent and the control module of particular API and runtime framework.

The AP-Ag and Mid-Ag are designed and implemented as AIoT agent. The difference between these 2 agents is that middleware agents have IoT device's control program as the control module, while application agents have the task processing programs as the control module.

IoT devices generally offer API for programming its control logic, and it is necessary to program dedicated control module to invoke the API to realize desired action of the target IoT device. In AIoT agent's architecture, the *Dedicated Control Module* (DCM) for agent invokes the API to perform device actions, as well as to receive responses to the agent. Through the control module, agent performs actions to solve problems.

In the agent part, there are 3 kinds of knowledge, namely, *Problem Solving Knowledge* (PSK), *Embodiment Knowledge* (EK), and *Cooperation Knowledge* (CK). The knowledge in AIoT agent is represented as a set of if-then rules, and ontologies which define the primitive vocabulary used in the rules. The knowledge is processed by rule-based inference engine.

PSK is a rule set and primitive vocabulary which define the following items:

- Actions to solve problems using the IoT device
- Conditions to recognize particular events

For example, if certain event is detected in control module part, and the event data is raised from the control module to the agent part, also if there is a rule which has the received event as the condition, then the rule is fired and the designated action is executed by sending command to the control module. Control module sends back the response of the executed command, and the response possibly fires other rules, and so forth.

In order to properly process task by avoiding conflicts in IoT device control, EK represents the following items as a set of rules and the self-embodiment ontology which defines the primitive vocabulary and records current IoT device's state:

- Physical characteristics on resource constraints of controlling IoT device
- Partial procedural knowledge on control module about the effect of each device action

Unlike ordinary software processes, there are number of constrained resources in physical IoT device, and individual actions are effected by one another even in a single IoT device. For example, it is impossible to perform sensing task of a specific point of place, while moving to another location to process another task. AIoT agent needs to be aware of the condition and occupation of resources, by taking into account the composite structure and characteristics of the IoT device. EK is defined to provide such awareness for agents. In case of AP-Ag, the EK is about the task processing programs and its constrained property to avoid conflicts of actions in runtime.

CK is a rule set and primitive vocabulary to cooperate with other agents to solve problems together, which defines the following:

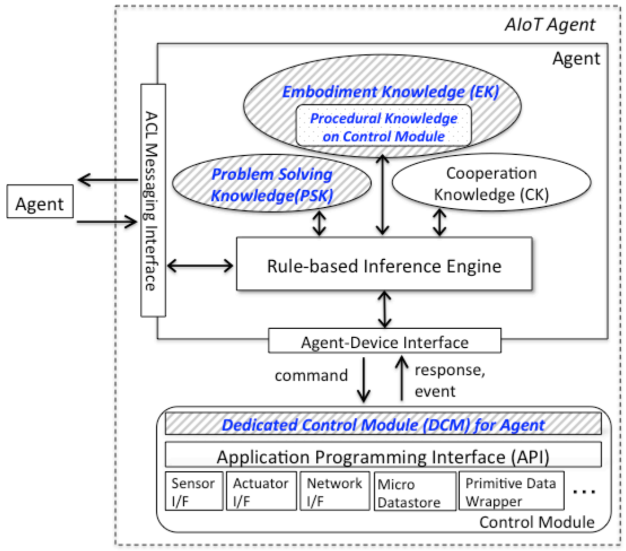

AIoT Agent
Agent
Embodiment Knowledge (EK)
Procedural Knowledge on Control Module
Problem Solving Knowledge(PSK)
Cooperation Knowledge (CK)
ACL Messaging Interface
Agent
Rule-based Inference Engine
Agent-Device Interface
command
response, event
Dedicated Control Module (DCM) for Agent
Application Programming Interface (API)
Sensor I/F
Actuator I/F
Network I/F
Micro Datastore
Primitive Data Wrapper
...
Control Module

Fig. 3 Architecture of AIoT agent (Mid-Ag or AP-Ag) that composes AIoT application.

- AIoT organization scheme to construct AIoT application
- AIoT re-organization scheme to adapt the AIoT application system towards requirement changes

As mentioned earlier, in AIoT architecture, agents are organized in task-oriented manner, and adaptively re-organized in response to the various changes. By this means, AIoT agent's knowledge is designed and described based on the characteristics of IoT device, and application's requirement for the devices. The default items are provided as knowledge template by AIoT middleware.

In addition to autonomous capability of AIoT agents, since the control module, AP-Ag, and Mid-Ag are architecturally separated, it requires less effort to modify each part of the AIoT devices against fundamental changes compared to the existing architecture [5] which targets construction of IoT applications.

ii) Agentification process of IoT device

The design and implementation process of Mid-Ag is the "agentification" process to develop AIoT device. Agentification is a process to realize the agent-based control of IoT device based on the agent's knowledge, as well as to cooperate with the other agents. Agentification provides IoT devices with the software agent's capability. The heterogeneity is resolved by agentification process to develop heterogeneous (function, communication protocol) IoT devices into homogeneous AIoT devices controlled by AIoT agents. Following steps are the process

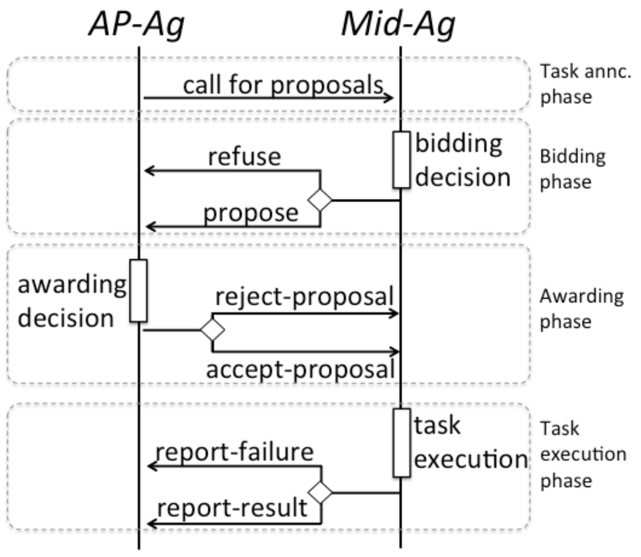


Fig. 4 Contract conclusion process of AIoT organization scheme between manager and contractor. (in this case, manager = AP-Ag, contractor=Mid-Ag).

of agentification. Developer of AIoT agent needs to implement the hatching part of the architecture in Fig. 3. The agentification involves the design and implementation of blue, hatched part of the AIoT agent.

Agentification Steps of IoT Device

Step 1. *Analysis of target IoT device, and design of the device action for problem solving*: The first step of agentification is to determine and list the actions of IoT device, by analyzing the physical characteristics and provided API of the IoT device. The determined behavior becomes a primitive set of Mid-Ag's actions in solving problem.

Step 2. *Design and implementation of DCM (Dedicated Control Module) to handle events and actions*: This step is to design and implement the DCM for handling command and response, as well as the event recognition. DCM for agent is designed based on the designed action of IoT device. To realize the targeted action, the DCM invokes physical actuator or sensor through APIs. The response is sent back to the agent as required by the command (result of actuation, sensed value from sensor, etc.). DCM also handles event detection, i.e. certain pattern of sensed value, detected error, etc. The event needs to be recognized and handled by the agent by referring to its knowledge.

Step 3. *Design and implementation of PSK and EK*: This step is to design and implement the knowledge as rule sets and ontology to handle the raised event and to control IoT device to solve problems. PSK rule sets determine the way to solve problems by actuating IoT device. The actuation is performed by referring to the EK of IoT device which represents the structure and characteristics of the device itself. In this step, common template for writing PSK and EK is given by AIoT middleware as agent templates, and CK is commonly given as common rule sets.

3.4 AIoT Organization Scheme for autonomous IoT system composition

AIoT organization scheme is a cooperation process of AIoT agents to compose AIoT application for executing assigned tasks by users. It is designed by extending Contract Net Protocol [19] especially for IoT device characteristics. The rest of this subsection explains the details of this scheme.

i) Cooperation flow of AIoT organization scheme

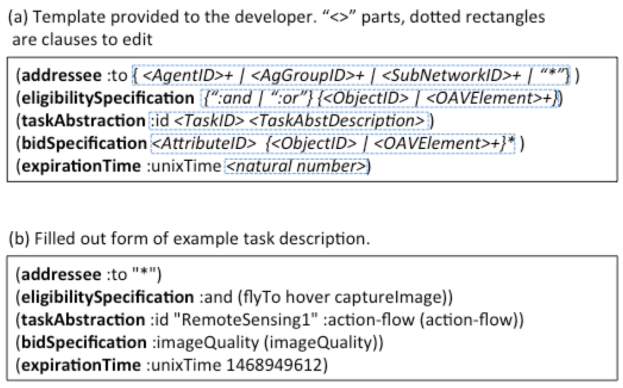
The sequence of cooperation flow is shown in Fig. 4. Fig. 4 is an interaction flow of task assigned manager (e.g. AP-Ag) to conclude a contract with a contractor (e.g. Mid-Ag) that executes the task. In this case, AP-Ag is the manager, and Mid-Ag is the contractor. However, the knowledge to realize this cooperation protocol is embedded in all the AIoT agent, thus both of AP-Ag and Mid-Ag can become manager to organize agents as described in Section 3.2.

- *Task announcement phase*

  At the beginning of contract conclusion process, manager sends *call for proposals* message to the prospective contractors. If applicable, manager divide a task into multiple subtasks, and announces one of the subtasks. This phase is called *task announcement*. The message includes the task name, task abstract, eligibility to become a contractor, and the expiration time of the announced task. Manager can choose the scope of task announcement, such as point to point, multicast, and broadcast.

- *Bidding phase*

  The contractor makes decision whether or not to bid on the announced task, based on the knowledge. This phase is called *bidding*. The contractor sends propose message to bid on the task, or sends refuse message not to bid on the task.


(a) Template provided to the developer. "<>" parts, dotted rectangles are clauses to edit

```
(addressee :to { <AgentID>+ | <AgGroupID>+ | <SubNetworkID>+ | "*"} )
(eligibilitySpecification {":and | ":or"} {<ObjectID> | <OAVElement>+})
(taskAbstraction :id <TaskID> <TaskAbstDescription> )
(bidSpecification <AttributeID> {<ObjectID> | <OAVElement>+}* )
(expirationTime :unixTime <natural number>)
```

(b) Filled out form of example task description.

```
(addressee :to "*")
(eligibilitySpecification :and (flyTo hover captureImage))
(taskAbstraction :id "RemoteSensing1" :action-flow (action-flow))
(bidSpecification :imageQuality (imageQuality))
(expirationTime :unixTime 1468949612)
```

Fig. 5 Example of task knowledge description used in task announcement.

In our proposal, the refuse message in bidding phase is introduced as an extension to conventional CNP, in order for manager agents to recognize the other agents' conditions, such as number of prospective agents in agent workplace, which helps the manager agents in various ways.

When Mid-Ag is a contractor, it is necessary to systematically examine the function and condition of the AIoT device by referring to the self-knowledge. We propose *embodied self-recognition method*, and *extended self-recognition method* to systematically evaluate the condition of AIoT device. The checking method of self-eligibility is explained later in this subsection.

- *Awarding phase*

  It is possible for the manager to receive multiple proposals on announced task. After the expiration time written in the task announcement message, the manager chooses one of the contractors by examining the contents of the proposal messages sent from the prospective contractors. This phase is called *awarding*. The manager sends *accept-proposal* message to award the task to the contractor, and *refuse-proposal* message to the rest of the contractors which sent the *propose* messages. When the *accept-proposal* message is received by the contractor, contract is properly concluded between the manager and contractor.

  In our proposal, the *reject-proposal* message is introduced as an extension to the conventional CNP, in order for contractors to recognize the situation faster to move onto the other tasks.

- *Task execution phase*

  The task awarded contractor executes the assigned task, and sends a report to the manager. This phase is called *task execution*. When the report is sent to the manager, the contract between manager and contractor is dissolved.

The manager derives the result of the task based on the report. In case that the manager divides tasks into subtasks, the manager concludes contracts and receives the report of each task. The manager integrates the reports of subtasks, to derive the result of the task.

By the contract conclusion scheme mentioned above, in AP layer, AP-Ag is able to organize agent organization which corresponds to service requirement. In Mid layer, Mid-Ag is able to organize Mid-Ags to deal with more versatile tasks.

ii) Knowledge on task and its announcement

The knowledge representation scheme and task description example is shown in Fig. 5. Full task knowledge representation scheme is shown in Fig. 26 of the appendix. AIoT architecture and middleware employs Object-Attribute-Value (OAV) format to represent knowledge. The example of the filled out form of task knowledge description is remote sensing task which is trying to recruit an agentified autonomous unmanned aerial vehicle (UAV). The UAV is expected to have functions to fly to specified point, as well as to capture image using camera.

addressee clause represents the agents to send task announcement message. Options are: point to point (*AgentID*), multicast (*AgGroupID*, *SubNetworkID*), or broadcast (*). The example employs broadcast for its task announcement scope.

eligibilitySpecification represents the required eligibility of contractors. There are *:and* and *:or* attribute that represents the condition of eligibility. The value part (*ObjectID* and *OAVElement*) is a list of keywords and OAV clause which represent the function and condition. These values are matched with the description of contractor's eligibility knowledge in bidding phase. In the example, *:and* attribute is employed for a set of cunctional eligibilities, which are *flyTo*, *hover*, *captureImage* functions are all required for the task.

taskAbstraction clause represents the abstracted description of a task. The *:id* attribute and *TaskID* represents the ID of the task. The *TaskAbstDescription* represents the content of the task. It is expected to include some of the action primitives for contractor to examine the task content. The *:action-flow* attribute and *action-flow* object ID represents the user-defined description of the task. In the contractor side, this description needs to be interpretable. Manager can send more information about abstracted task, e.g. the attributes associated with the *action-flow* object.

bidSpecification represents the requirement of content expected in bidding message. The attribute and values need to be specified in this clause. In the example, the image quality is set as the bidding information. This information is used by manager to compare and decide the contractors to assign the task. This description also need to be interpretable by the contractors to include proper information in bidding message. Manager can also send more information about bid specification, e.g. attributes about image quality such as image size, frame rate, etc.

expirationTime clause represents the time of expiration for contractors to bid on the task. We mainly employ Unix time to represent this time.

iii) Embodied Self-Recognition Method

As mentioned earlier, bidding phase is extended to systematically examine the function and condition of Mid-Ag's AIoT device. It is necessary to develop systematical method to validate the function and condition of IoT device, because of the following reasons:

- **To mitigate development complexity and burden**: Heterogeneity of IoT device makes it complicated to develop validation procedures individually for each function of each device
- **To avoid runtime conflict**: Since IoT device operates in dynamical physical environment, it is insufficient to refer to the static description about the IoT device

Fig. 6 shows the overview of embodied self-recognition process in Mid-Ag. In order to evaluate the feasibility of announced task, it is necessary to monitor the conditions of resources based on the physical architecture and constraints. The embodied self-recognition process is as follows:

Steps of Embodied Self-recognition Process

Step 1. *Matching*: After receiving task announcement and requirement in the eligibility specification clause, Mid-Ag matches its own eligibility and lists the actions required to realize the eligibility.

Step 2. *Sensing*: If the listed actions require physical actuation and/or sensing, the Mid-Ag refers to the conditions associated with the physical components which provide the listed actions.

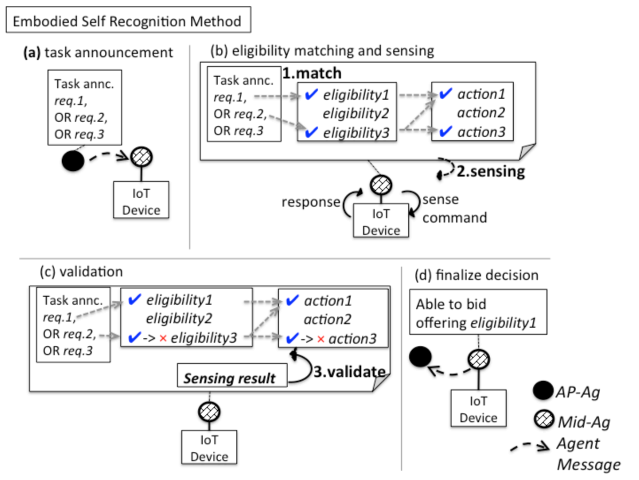

Embodied Self Recognition Method
(a) task announcement
Task annc. req.1, OR req.2, OR req.3
IoT Device
(b) eligibility matching and sensing
Task annc. req.1, OR req.2, OR req.3
1.match
eligibility1 eligibility2 eligibility3
action1 action2 action3
2.sensing
response
sense command
IoT Device
(c) validation
Task annc. req.1, OR req.2, OR req.3
eligibility1 eligibility2 eligibility3
action1 action2 action3
Sensing result
3.validate
IoT Device
(d) finalize decision
Able to bid offering eligibility1
IoT Device
AP-Ag
Mid-Ag
Agent Message

Fig. 6 AIoT Device's Embodied Self-Recognition Method.

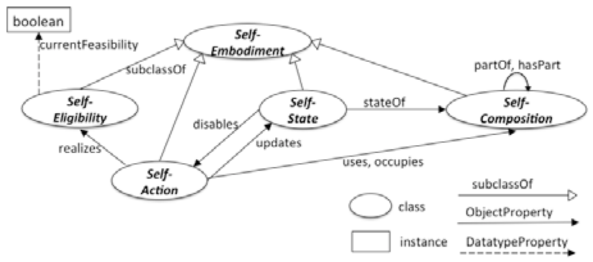


Fig. 7 Top-level Class Definition of Self-Embodiment Knowledge Ontology for IoT Devices.

Step 3. *Validation*: If the required physical parts are unavailable (e.g. occupied with other process, lacking resource, etc.), the Mid-Ag removes the unavailable action from the list. If the action availability still satisfies the required eligibility, Mid-Ag is able to bid on the announced task.

In order to realize the embodied self-recognition of IoT device, we introduced self-embodiment ontology in EK (embodiment knowledge). The ontology defines the relationship among the IoT device's eligibilities, actions, physical resources and the condition of the resources. The ontology systematically supports the *matching*, *sensing* and *validation* steps mentioned above.

Fig. 7 shows the class definition of the embodiment ontology. There are 5 classes in the ontology, namely, *Self-Embodiment*, *Self-Eligibility*, *Self-Action*, *Self-State*, and *Self-Composition*. Top level class is the *Self-Embodiment*, and other 4 classes are the subclasses.

Self-Eligibility class has the keywords of eligibility as instances, which match the eligibility specification as shown in the example of Fig. 5. *Self-Action* class has the keywords of actions as instances which match the control

$$\text{Self-Eligibility}(?se) \wedge \text{realizes}(?sa, ?se) \wedge \sim\text{disables}(?ss, ?sa)$$
$$\wedge\ \text{uses}(?sa, ?sc) \wedge \sim\text{occupies}(?sa, ?sc)$$
$$\rightarrow \text{currentFeasibility}(?se, \text{true})$$

Fig. 8 Description Example of Self-Eligibility Validation Rule, by Taking into Account the States of Corresponding Actions for the Eligibility.

procedure name which is executed in the IoT device's DCM (dedicated control module). *Self-Composition* class has the keywords of the IoT device's composing parts, which physically realize the action instances under the *Self-Action* class, e.g. motors, battery, sensors, etc. *Self-State* class has the keywords of conditions as instances, which are associated with actual variables in the DCM.

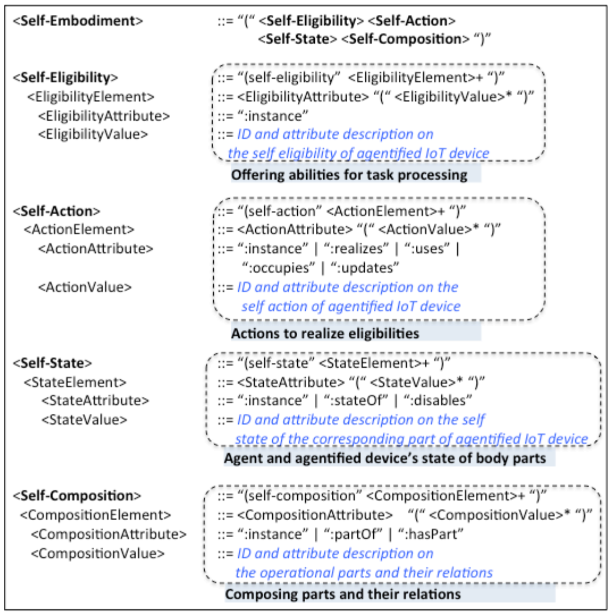
The properties are representing the relationships among the classes, and these properties are utilized to derive the feasibility of the announced task. Fig. 8 is a description example of one of the rules to validate the eligibility instances. It functions as: "If, for a self-eligibility instance *?se*, self-action instance(s) *?sa* realizes self-eligibility instance *?se*, and self-state instance(s) *?ss* does not disable self-action instance(s) *?sa*, and self-action instance(s) *?sa* uses self-composition instance(s) *?sc*, and self-action instance(s) *?sa* does not occupy self-composition instance(s) *?sc*, then set current feasibility of the self-eligibility instance *?se* as *true*."

The *disables* property and *occupies* property are inserted and removed according to the sensing in the IoT device by Mid-Ag. The proposed ontology helps developers, administrators and agents to recognize self-recognition to properly bid on the task.

In proposed AIoT architecture and middleware, the ontology and rules are represented in OAV description. Fig. 9 shows the self-embodiment knowledge representation scheme, and Fig. 10 shows the task description example. Full description of representation scheme is shown in Fig. 27.

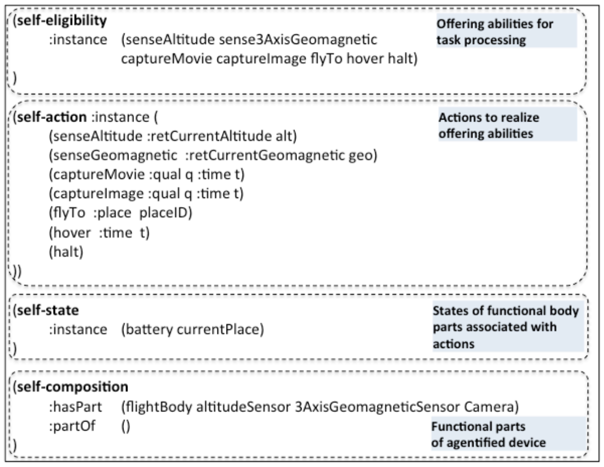
The description example is also about the agentified UAV with camera, matching with the target IoT device in the example of task description. The *self-eligibility* clause has the list of keywords representing the eligibilities, which matches the *eligibilitySpecification* in Fig. 5. The *self-action* clause has the list of keywords representing the actions that realize the eligibilities. Individual clause of *self-action* has attributes of the actions as objects, and concrete values are associated with the objects IDs.

The *self-state* clause has the keywords representing the state of physical component. In the example, it has *battery* status, and *currentPlace*. These are object IDs, and



```
<Self-Embodiment>              ::= "(" <Self-Eligibility> <Self-Action>
                                      <Self-State> <Self-Composition> ")"

<Self-Eligibility>             ::= "(self-eligibility"  <EligibilityElement>+ ")"
  <EligibilityElement>         ::= <EligibilityAttribute> "(" <EligibilityValue>* ")"
    <EligibilityAttribute>     ::= ":instance"
    <EligibilityValue>         ::= ID and attribute description on
                                   the self eligibility of agentified IoT device
                                   Offering abilities for task processing

<Self-Action>                  ::= "(self-action" <ActionElement>+ ")"
  <ActionElement>              ::= <ActionAttribute> "(" <ActionValue>* ")"
    <ActionAttribute>          ::= ":instance" | ":realizes" | ":uses" |
                                     ":occupies" | ":updates"
    <ActionValue>              ::= ID and attribute description on the
                                     self action of agentified IoT device
                                   Actions to realize eligibilities

<Self-State>                   ::= "(self-state" <StateElement>+ ")"
  <StateElement>               ::= <StateAttribute> "(" <StateValue>* ")"
    <StateAttribute>           ::= ":instance" | ":stateOf" | ":disables"
    <StateValue>               ::= ID and attribute description on the self
                                     state of the corresponding part of agentified IoT device
                                  Agent and agentified device's state of body parts

<Self-Composition>             ::= "(self-composition" <CompositionElement>+ ")"
  <CompositionElement>         ::= <CompositionAttribute>   "(" <CompositionValue>* ")"
    <CompositionAttribute>     ::= ":instance" | ":partOf" | ":hasPart"
    <CompositionValue>         ::= ID and attribute description on
                                     the operational parts and their relations
                                  Composing parts and their relations
```

Fig. 9 Part of Self-Embodiment Knowledge Description Scheme that Represents the Self-Embodiment Ontology.



```
(self-eligibility                                              Offering abilities for
      :instance   (senseAltitude sense3AxisGeomagnetic         task processing
                   captureMovie captureImage flyTo hover halt)
)

(self-action :instance (                                       Actions to realize
      (senseAltitude :retCurrentAltitude alt)                  offering abilities
      (senseGeomagnetic  :retCurrentGeomagnetic geo)
      (captureMovie :qual q :time t)
      (captureImage :qual q :time t)
      (flyTo  :place  placeID)
      (hover  :time  t)
      (halt)
))

(self-state                                                    States of functional body
      :instance   (battery currentPlace)                       parts associated with
)                                                              actions

(self-composition
      :hasPart    (flightBody altitudeSensor 3AxisGeomagneticSensor Camera)
      :partOf     ()                                           Functional parts
)                                                              of agentified device
```

Fig. 10 Example of Filled-out Form of Self-Embodiment Knowledge Description which Represents an agentified UAV Capable of Performing Remote Sensing Task (as in Fig.5).

concrete values acquired from the IoT device through DCM, are associate with the IDs.

The *self-composition* clause has the composing parts of the IoT device. In the example, the sensors and flight body are represented as the keywords. For example, *flyTo* action occupies the *flightBody*, but *sense3AxisGeomagnetics*

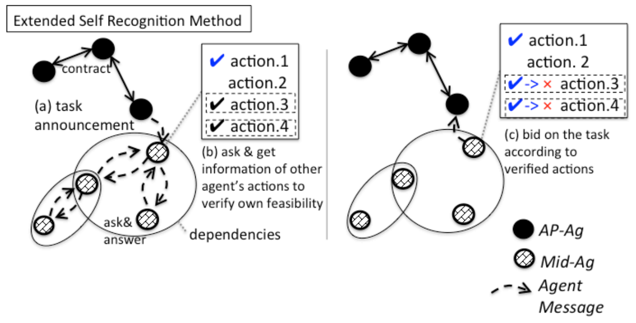


Fig. 11 Overview of Extended Self-Recognition Method.

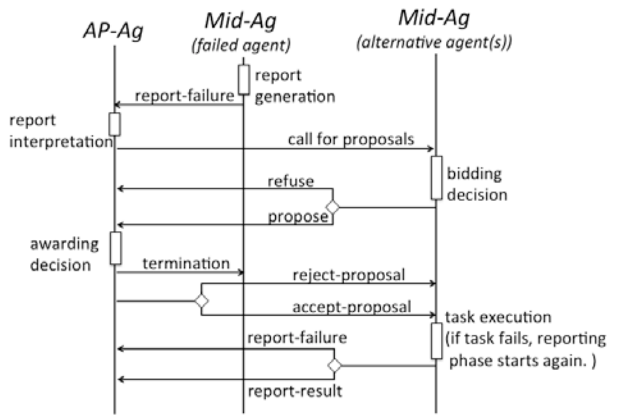


Fig. 12 Re-organization process among a manager (Ag-AP), failed contractor (Mid-Ag), and prospective contractor(s) (Mid-Ag).

action does not occupy the *3AxisGeomagneticsSensor*, hence and it is not possible for multiple tasks to fly the UAV simultaneously, but possible for sensors because sensor value is sharable in parallel manner.

iv) Extended Self-Recognition Method

In addition to the embodied self-recognition method, we propose extended self-recognition method in order to systematically examine the conditions of conjugated agents of task.

It is frequent for Mid-Ag to perform an action by using the other agent's resources. It is necessary to develop systematical method to validate the function and condition of conjugated IoT devices, because of the following reasons:

- **To avoid runtime conflict in cooperation**:

  When a Mid-Ag is fully available, it usually considers that it is available for task processing. However, if cooperating agent is occupied with other task, task execution fails because of the cooperating agents.

- **To avoid agent messaging overload**:

  Without systematical support to examine conjugated agent's condition, it is difficult to code the message flow for each action with the other agents. It induces agent message overload among agents, because without systematical method, there is no clear end to exchanging messages to monitor other agent's condition.

Fig. 11 shows the overview of extended self-recognition process in Mid-Ag. Extended self-recognition method is an extension of proposed embodied self-recognition method. In the embodied self-recognition method, when a Mid-Ag receives a task announcement, it lists the matching eligibilities and the realizing actions of the eligibilities. If an action utilizes other agent's action, the Mid-Ag sends *information-request* message, and receives *information-reply* message to check the availability. The request and reply message are triggered when there are agent IDs with the referred agent's object ID in the following clauses:

- **Self-Eligibility**: Mid-Ag sends the referred agent the referred eligibility ID, and the asked agent replies a binary value representing the availability of the asked eligibility
- **Self-Action**: Mid-Ag sends the referred agent the referred action ID, and the asked agent replies a binary value representing the availability of the asked action
- **Self-State**: Mid-Ag sends the referred agent the referred state ID, and the asked agent replies the associated values to the asked state ID.
- **Self-Composition**: Mid-Ag sends the referred agent the referred parts ID, and the asked agent replies the associated values to the asked parts ID.

The relationship among AIoT agents which have the other agent's ID in the embodiment knowledge with each other, is called conjugate relation. The agents in conjugate relation works tightly together to solve problems, and frequently share information with each other. Concrete example of the knowledge description on extended self-recognition is given in Section 4.1 with the implemented AIoT device.

3.5 Resilient Service Re-organization Scheme for IoT Applications

The condition of task processing organization changes (lowering task processing speed, suspension of action, etc.) due to the fluctuation in the individual agents and IoT devices' conditions (CPU load, physical failure, etc.), as well as the condition of environment (change of weather, appearance of unexpected object, etc.).

In order to deal with the changes to sustain the task as an organization, we propose the real-time replacement and tuning of agents in organization in AIoT re-organization scheme. To realize the re-organization of agent, manager needs to have CK (cooperation knowledge) to interpret the events and requests that arise in the agent organization, and to tune and replace the agents based on the interpretation.

A manager needs to have various knowledge to control the contractors, concretely, the knowledge on error report interpretation, task description modification, agent selection, agent activation, agent termination, etc. Common CK on these actions are embedded into each AIoT agent participating the organization by AIoT middleware.

Fig. 12 shows the proposed AIoT re-organization scheme's flow which extends the AIoT agent organization scheme. In case that a contractor (e.g. Mid-Ag) fails at the assigned task, the agent sends the report to the manager (e.g. AP-Ag). The report includes the task ID, detected event and device conditions, etc.

After receiving the report, the manager interprets the report, then starts AIoT organization of the reported failed task (subtask). If necessary, the task description is modified by the manager before starting the AIoT organization.

If there is an alternative proposal among the *propose* messages which can handle the failed task, the manager terminates the failed Mid-Ag, and simultaneously award a task to the matched agent. If the manager cannot find the alternative agent, the organization is no longer able to continue the task.

AIoT re-organization is not only about replacing agent, but also tuning of participating agents in organization, thus, the report is not necessarily a failure report, it can be the processing time report of task. If the processing time is considered insufficient, manager can either replace the agent with others, or change the parameters of the task.

The re-organization protocol can be used both in AP-Ag and Mid-Ag, to organize agents in both in AP layer and Mid layer. Concrete example of the re-organization and task modification is given in Section 4.3.

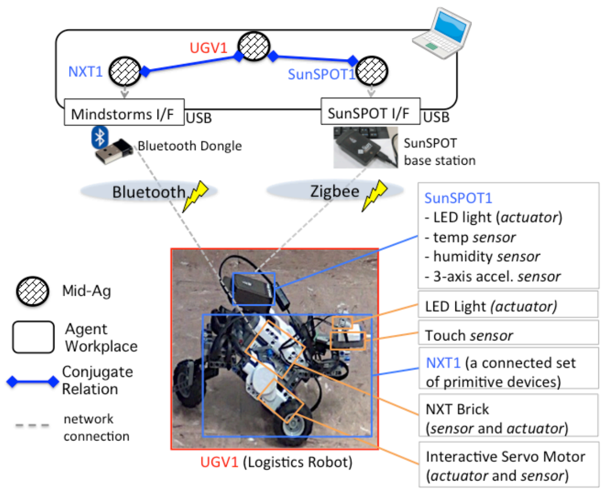


Fig. 13 Agent and device configuration of logistics robot (UGV1) controlled by Mid-Ags.

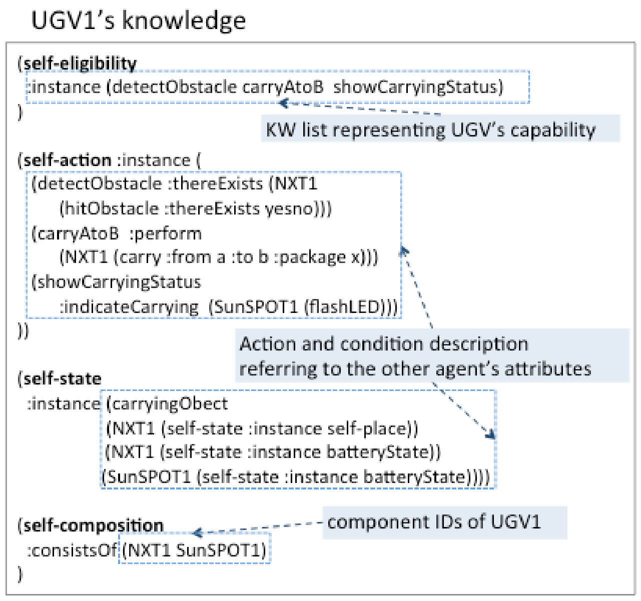
4. Implementation and Experiment of AIoT Logistics Application for AIoT Architecture Evaluation

In order to confirm the effectiveness of proposed AIoT architecture and AIoT middleware, we have implemented an autonomous logistics [20] application using simplified small robots with attached sensors.

In this section, we confirm the effect of AIoT device agentification, AIoT organization and re-organization scheme to realize autonomic and resilient service provisioning in IoT system. The simplified logistics system is organized by agentified robots of logistics and an application agent. We also performed architecture-level analysis on the organized AIoT logistics system, based on the comparison with the conventional architecture-based implementation [5] of the equivalent logistics application.

4.1 Configuration of Implemented Experimental System using AIoT Devices

We have implemented the autonomous logistics application using repository-based multi-agent framework, ADIPS [21][22]. The AIoT agents are implemented as software agents of the ADIPS framework. Each ADIPS agent has a built-in inference engine, and runs as an



UGV1's knowledge

```
(self-eligibility
 :instance (detectObstacle carryAtoB  showCarryingStatus)
)
(self-action :instance (
 (detectObstacle :thereExists (NXT1
     (hitObstacle :thereExists yesno)))
 (carryAtoB  :perform
     (NXT1 (carry :from a :to b :package x)))
 (showCarryingStatus
     :indicateCarrying  (SunSPOT1 (flashLED)))
))
(self-state
 :instance (carryingObect
           (NXT1 (self-state :instance self-place))
           (NXT1 (self-state :instance batteryState))
           (SunSPOT1 (self-state :instance batteryState))))

(self-composition
 :consistsOf (NXT1 SunSPOT1)
)
```

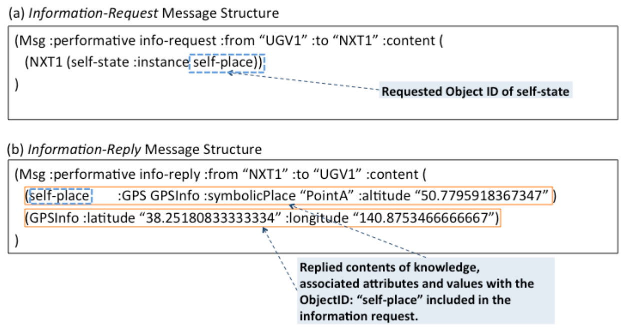
KW list representing UGV's capability
Action and condition description referring to the other agent's attributes
component IDs of UGV1

Fig. 14 EK of UGV1 in OAV form.

independent computing process. The control knowledge is described in rules and embedded in the agents, so are the proposed 3 kinds of knowledge, PSK, EK and CK. The ADIPS agent has an interface function that interacts with a procedural program, and the function is used to realize agentification of IoT device. The rest of this subsection describes the implementation of AIoT agents and IoT devices in autonomous logistics application using ADIPS framework.

Fig. 13 shows the configuration of logistics robot and its controlling agent. This robot consists of two IoT devices, which are SunSPOT (multiple functional sensor) [23] and LEGO Mindstorms NXT intelligent kit [24]. These two devices are controlled individually via different APIs. Logistics robot has several sensors and actuators, such as LED light, touch sensor, servomotors, etc. The control logic of each part of NXT is programmed in the agent's knowledge as EK in if-then rule sets. SunSPOT also equips multiple functions, and the control logic is likewise programmed in the agent's knowledge. The Mid-Ags controlling Lego Mindstorms NXT and SunSPOT have PSK to perform primitive actions to solve problem using control logic in EK, and CK for agent organization and re-organization.

Fig. 14 shows the partial EK description on UGV1 agent. As described in Section 3, as a Mid-Ag, UGV1 has self-eligibility, self-action, self-state, and self-composition knowledge in part of EK. These attributes are referred by rule sets in CK, PSK, and EK. For example, as shown in self-eligibility clause in Fig. 14, UGV1 is able to detect



(a) *Information-Request* Message Structure

```
(Msg :performative info-request :from "UGV1" :to "NXT1" :content (
  (NXT1 (self-state :instance self-place))
)
```

(b) *Information-Reply* Message Structure

```
(Msg :performative info-reply :from "NXT1" :to "UGV1" :content (
  (self-place      :GPS GPSInfo :symbolicPlace "PointA" :altitude "50.7795918367347" )
  (GPSInfo :latitude "38.25180833333334" :longitude "140.8753466666667")
)
```

Requested Object ID of self-state
Replied contents of knowledge, associated attributes and values with the ObjectID: "self-place" included in the information request.

Fig. 15 Information Request and Reply Message Format of Agent Status Information based on Proposed Knowledge Representation.

obstacle, carry things from point to point, and show carrying status of UGV itself. Also, conjugated agents, such as NXT1 agent and SunSPOT1 agent, are referred in self-action, self-state, and self-composition clauses, because UGV1 physically consists of NXT1 and SunSPOT1. The composite property of UGV1 is represented by these references.

Although the UGV1 physically does not have functionality to physically carry package from point to point, the agent emulates the package handover among UGVs in order to demonstrate autonomous logistics based on AIoT agents. Only the information of package is exchanged among agents. The UGV carrying a package lights the LED built in a SunSPOT, therefore it is possible to visually confirm the handover of a package in experiment.

As shown in Fig. 15, using the agent ID and property ID, agents are able to exchange information request and reply message to monitor individual state as a UGV. Likewise, SunSPOT1 agent and NXT1 agent has self-embodiment knowledge representing the functions and states. The handover of package is realized by an information exchange among two logistics robots when two robots are physically near to each other.

As shown in Fig. 15, using the agent name and property name, agents are able to exchange information request and reply message to monitor individual state as a UGV. Likewise, SunSPOT1 agent and NXT1 agent has self-embodiment knowledge representing the functions and states. Since UGV1, NXT1 and SunSPOT1 agents are physically coupled together, NXT1 and SunSPOT1 agents are designed not to respond to task announcement message, because UGV1 agent represents these agent's and controls them.

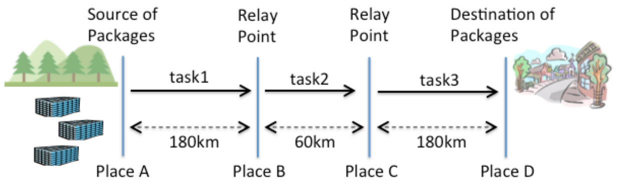
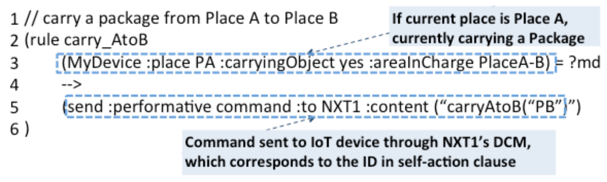


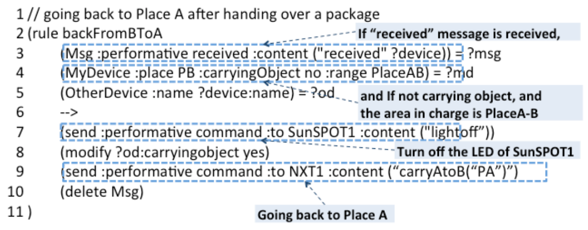
Fig. 16 Overview of autonomous logistics task for AIoT agents.



```
// carry a package from Place A to Place B
(rule carry_AtoB
     (MyDevice :place PA :carryingObject yes :areaInCharge PlaceA-B) = ?md
     -->
     (send :performative command :to NXT1 :content ("carryAtoB("PB")")
)
```

If current place is Place A, currently carrying a Package
Command sent to IoT device through NXT1's DCM, which corresponds to the ID in self-action clause

Fig. 17 Description example of PSK representing logistics action of UGV, to carry a package to Place B from Place A after picking up a package from the source.



```
// going back to Place A after handing over a package
(rule backFromBToA
     (Msg :performative received :content ("received" ?device)) = ?msg
     (MyDevice :place PB :carryingObject no :range PlaceAB) = ?md
     (OtherDevice :name ?device:name) = ?od
     -->
     (send :performative command :to SunSPOT1 :content ("lightoff"))
     (modify ?od:carryingobject yes)
     (send :performative command :to NXT1 :content ("carryAtoB("PA")")
     (delete Msg)
)
```

If "received" message is received,
and If not carrying object, and the area in charge is PlaceA-B
Turn off the LED of SunSPOT1
Going back to Place A

Fig. 18 Description example of PSK representing logistics action of UGV to go back to the source of package after handing over a package at Place B.

We have also introduced an UAV, Parrot AR. Drone [25] as a logistics robot, which is implemented as AIoT device through the proposed agentification. The agent of the UAV, UAV agent also has the knowledge on logistics application similar to the UGV agent. The difference between UGV and UAV agent is that UAV can fly to a specified point with carrying a package, so that UAV can carry package over an obstacle, unlike UGV.

In the next section, we explain an autonomous logistics application using multiple AIoT devices, which are the UGV, and UAV mentioned above.

4.2 Experiment 1: AIoT Organization of Autonomous Logistics Application among Heterogeneous Devices

Using the AIoT agent and device implemented as in last subsection, experiment 1 examines the effect of following proposals:

- **S1)** Agentification of IoT device to AIoT device
- **S2**) AIoT Organization Scheme for Autonomic AIoT Application Composition

We evaluate these proposals by a case study of autonomous logistics using unmanned vehicles. Following sub-subsections explains the given task to AG-Ag, the organization process of autonomous logistics application, the result of logistics experiments, and scalability analysis of the organization.

i) Given Autonomous Logistics Task and AIoT Agents

The task is to carry packages from the source to the destination. Fig. 16 shows the overview of autonomous logistics task for AIoT agents. Place A is a source of packages, where construction is ongoing and packages of timbers are continuously generated, with the package ID corresponding to the order of shipping packages. Place D is a destination of the packages. The distance between Place A-B, Place B-C, Place C-D are 180km, 60km, and 180km, respectively. The distances are determined based on the typical physical constraints and characteristics of autonomous logistics vehicles [20].

The logistics task is divided into 3 tasks, i.e. task to carry package from Place A to Place B is task 1, Place B to Place C is task 2, and Place C to Place D is task 3. In each area, autonomous logistics robot handles the logistics of packages. The tasks are divided, because of the basic limitation of vehicles, such as fuel. In each area, at least one autonomous vehicle is required, hence three vehicles are required to accomplish the whole task.

We introduce three UGVs which have function to carry 1 package at a time from place to place. The UGVs are agentified and controlled by Mid-Ags. In addition to primitive control logic of UGV's physical parts, as PSK, the following actions of UGVs are introduced:

- Pickup a package
- Go to specified point
- Handover a package to the other logistics capable entity

The start and ending of the task is commanded by an AP-Ag of this autonomous logistics task, which is logistics agent. In each area, UGV picks up a package, goes to another point, and handover package to the other UGV. Each UGV repeats this flow of actions repeatedly.

At Place A, UGV picks up a package from the source, hence it does not receive package from other agent. At Place D, UGV handover package to the destination, hence it also does not need to handover package to the other UGV. The handover of package is taken place in the relay points of Place B and Place C.

Description example of problem solving knowledge is shown in Fig. 17 and Fig. 18. The rule of Fig. 17 is to carry a package from place A to place B. In line 2, carry_AtoB is a rule name, and the clause before “-->” is IF part, and clauses after the symbol is the THEN part. The rule reads as follows:

- **IF** my device (Mid-Ag’s device) is in Place A, not carrying object, and the area in charge is between Place A-B (line 3),
- **THEN** send the command of *carryAtoB(“PB”)* to move the device to Place B through NXT1 agent and its DCM (line 5).

The rule of Fig. 18 is to go back to Place A after handing over a package to other agent. The rule reads as follows:

- **IF** “*received*” message is received from *?device* (line 3), and my device is at place B, not carrying object, and the area in charge is Place A-B (line4),
- **THEN** send “*lightoff*” to SunSPOT1 agent to stop lighting LED, and then modify my status of carrying package to no, and send *goTo(“PA”)* command to NXT1 agent to go to Place A.

PSK, as well as the other knowledge’s rule description is written in this OAV form based rule language provided by ADIPS framework. The explained PSK on logistics is also embedded in UAV agent.

The knowledge description of task1 is shown in Fig. 19. The *eligibilitySpecificaiton* clause has the list of keywords representing the object carrying eligibilities, and the physical area in charge as “Place A-B” which means a logistics robot is in Place A-B and authorized to carry packages in the area. The *taskAbstraction* clause has a reference to logistics action flow, which is in *lgs-action-flow* clause from line 10. The *lgs-action-flow* clause is also sent to the prospective contractors. The *bidSpecification* clause has the required information from prospective contractors in bidding message, which are *loadWeight* and *fuelConsumption*. ADIPS agents can use this form of task description in the agent program file, and refer to these object IDs and values in the rules. The organization of autonomous logistics application is demonstrated and evaluated in the next sub-subsection.

```
1  (addressee :to "*")
2  (eligibilitySpecification :and (carryAtoB handOverPackage
3                                    pickUpPackage goTo
4                            (areaInCharge :value “PlaceA-B”) ))
5  (taskAbstraction :id ”task1" :action-flow (lgs-action-flow))
6  (bidSpecification :loadWeight loadWeight
7                      :fuelConsumption fuelConsumption)
8  (expirationTime :unixTime 1468949612)
9
10 (lgs-action-flow :flow (
11   (goTo :place "point A")
12   (pickUpPackage)
13   (goTo :place "point B")
14   (handOverPackage)
15 ))
```

Fig. 19 Knowledge description of autonomous logistics task (task1) in Logistics agent (AP-Ag), a part.

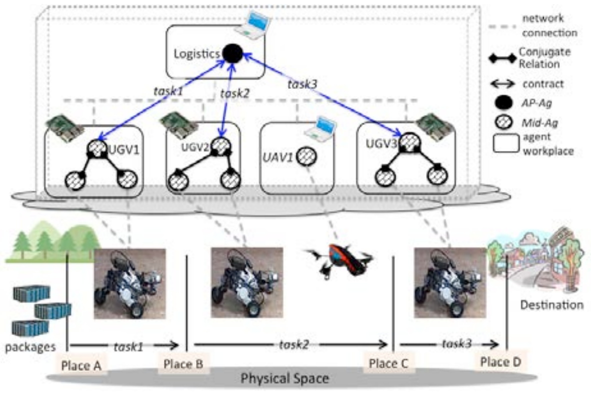


Fig. 20 Agent and device configuration of autonomous logistics application and contract conclusion result of task1, 2 and 3.

ii) Autonomic Organization of AIoT Logistics Application and Configuration of AIoT devices

In order to evaluate the feasibility of AIoT organization scheme, we performed the autonmic organization of AIoT logistics application logistics task explainer, and executed the autonomous logistics task among AIoT devices. The experiment configuration and contract conclusion result are shown in Fig. 20. We placed UGV1 in Place A-B, UGV2 and UAV in Place B-C, and UGV3 in Place C-D. The Mid-Ags of logistics robots are able to communicate through network via messages. UGV and UAV both have the ability to carry packages, and UAV has the advantage to carry things above obstacles. Logistics agent is an application agent that manages the task of carrying packages from Place A to D periodically.

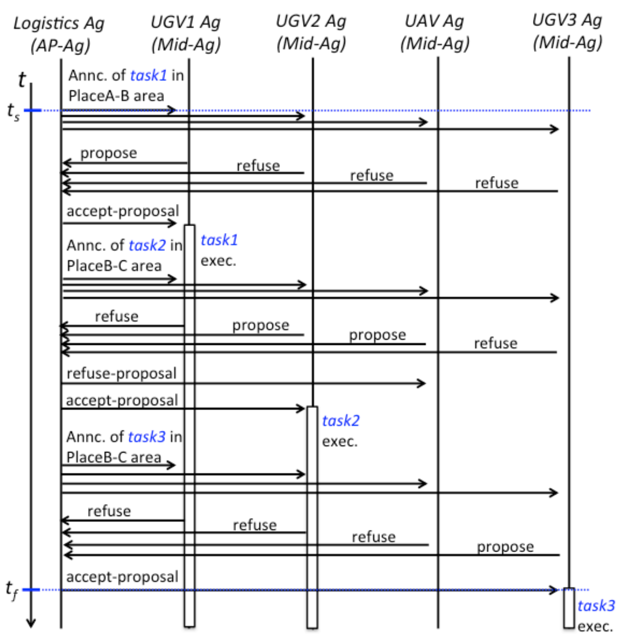


Fig. 21 Measured message flow and processing time measurement point of AIoT organization process to organize AIoT logistics application.

With these agents mentioned above, we started logistics experiment by the AIoT organization initiated by Logistics agent. We have measured the message flow among agents in AIoT organization, and measured the flow and processing time measured point as in Fig. 21. Logistics agent performs AIoT organization scheme three times, for task1, task2 and task3.

In the organization of task1, UGV1 is selected as a contractor. As soon as the task is awarded to UGV1, it starts carrying a package from Place A. Logistics agent makes the logistics start in Place A-B, because there is low possibility that UGV1 arrives at Place B before the selection of logistics entity in Place B-C. The other logistics robot also starts the logistics task right after the task assignment.

In the organization of task2, UGV2 is selected by Logistics agent. The selection among bidding proposal is dependent on the knowledge embedded in an Logistics agent. In this case, Logistics agent chooses UGV2 because it consumes less fuel for transportation. UGV3 is also selected like UGV1.

Fig. 22 shows the measured result of organized AIoT logistics application, with fifty contractors. Horizontal axis represents package IDs, which corresponds to the order of shipment from Place A. Vertical axis represents the order

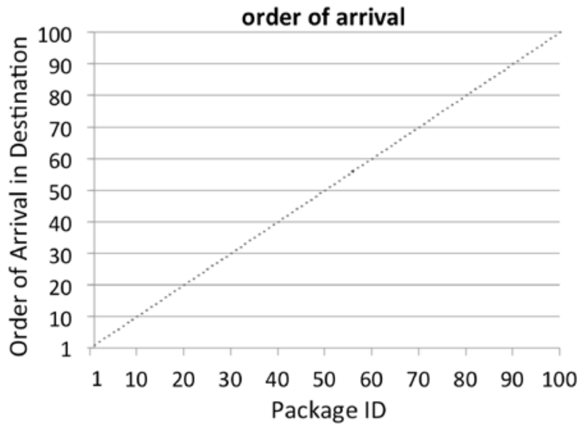


Fig. 22 The result of autonomous logistics, the order of package arrival.

of arrival at Place D. Therefore, ideal result without error is that all of the package IDs match the number of the order of arrival. As the result of logistics experiment using AIoT devices mentioned above, the degree of irregularity (*DoI*) was 0, as calculated below:

$$DoI = \frac{\sum_{i=1}^{N} |OA_i - OS_i|}{N}$$

where $N = 100$ which is the number of packages, OA is the order of arrivals, and OS is the order of shipment, i is the package ID. OA_i and OS_i are the order of arrival of package i, and the order of shipment of package i, respectively.

From this experimental result, we have confirmed the AIoT architecture based control’s feasibility of AIoT logistics in terms of the adequacy and accuracy. This is an effect of introduced PSK, as well as the agentification of IoT devices.

It is also confirmed that our proposal cleared the problem of the autonomic organization of AIoT application based on various requirements. Because of the agentification of logistics robots (Mid-Ag), as well as introducing Logistics agents (AP-Ag) that organizes the logistics application, heterogeneous AIoT devices are capable to cooperate together by organizing themselves based on AIoT organization scheme to accomplish a task in autonomic manner.

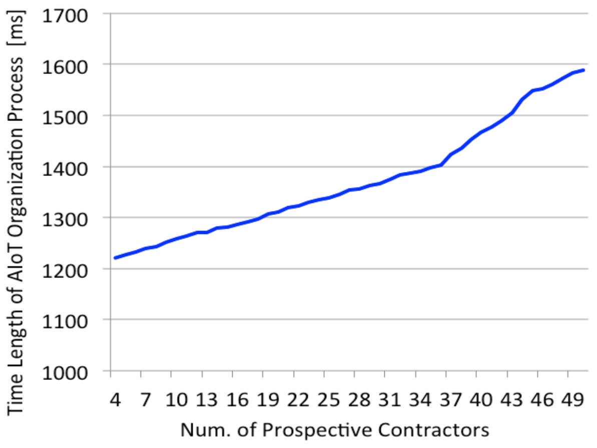


Fig. 23 Time Length of AIoT Application Organization (average of 1000-times trial runs.)

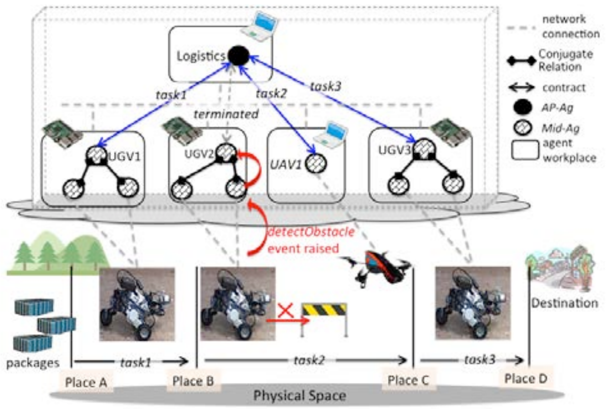


Fig. 24 Experimental result of AIoT re-organization scheme to sustain logistics application among UGVs and a UAV controlled by Mid-Ags.

iii) Processing Time of AIoT Organization with increasing number of AIoT devices

We have measured the processing time of AIoT organization, by increasing the number of contractor in emulation. In this measurement, we have used the logistics robots in emulation, in which AIoT agents are not physically connected to IoT devices, but the command and response from IoT device are emulated. In the experiments, the agent workplace and the agents are running on a single computer (CPU: Intel Core-i7 4GHz, Memory: 16GB, OS: macOS Sierra). The result of the simulation is shown in Fig. 23. Starting with four contractors (the same number of contractor in the logistics experiment above), we increased the number of contractors up to fifty. In each number of contractors, we have executed the AIoT organization one thousand times, and calculated the average time of organization process, which is the average time of organization process (*ATOP*).

In Fig. 23, horizontal axis represents the number of contractors participating in the AIoT organization, and vertical axis represents the average processing time of organization process as calculated below:

$$ATOP = \frac{\sum_{i=1}^{N} |t_{fi} - t_{si}|}{N}$$

where $N = 1000$ which is the number of trials, t_{fi} is the ending time of the organization as in Fig. 21, and t_{si} is the starting time of the organization.

From this experimental result, we have confirmed that even there are dozens of prospective contractors, the processing time of AIoT organization is in the order of a few seconds. It is fast enough for logistics application, because it is far less than the load time of carrying package in each area, hence the processing lag of AIoT organization does not disturb any interactions in the logistics application. Likewise, since the organization, service composition process are conventionally performed manually by an administrator, the order of seconds are considerably short.

4.3 Experiment 2: Resilient Adaptation in Autonomous Logistics Application

Experiment 2 examines the resilient adaptation capability of AIoT agent organization toward environmental change. In this experiment, among the proposed knowledge of AIoT agent, we focus on CK (cooperation knowledge) which is commonly introduced by proposed middleware. Therefore, the evaluated proposals is:

- **S3)** AIoT Re-organization scheme for Resilient AIoT Application Operation

In order to sustain the given task to AIoT agent organization, the organization needs to re-organize (replace agent(s), re-configure agent(s)) the member agent to address the problem caused by situational changes.

The task of this experiment is the same as experiment 1. It is to carry packages from Place A to Place D, and the path is divided into three areas (Place A-B, Place B-C, Place C-D), and there are logistics robots in each area.

As shown in Fig. 24, the difference between experiment 1 and 2 is the appeared obstacle in Place B-C. UGV2 receives a package at Place B and tries to go to Place C, however an obstacle appears between Place B and C to block UGV2. As described in Section 4.1, UGV2 has function to detect obstacle using its sensor. Fig. 24 also shows the experimental result of AIoT re-organization against the environmental change. As the result, through the AIoT re-organization scheme, task2 is awarded to UAV1 agent instead of UGV2 agent.

The measured message flow of the AIoT re-organization is shown in Fig. 25. After detecting an obstacle by the sensing result, UGV2 makes decision that it is no longer able to carry packages in the area Place B-C. Following the proposed AIoT re-organization scheme, UGV2 agent generates and sends a report to Logistics agent (manager). The report includes the abstracted task information, and UGV2's information about its self-state, as well as the detected event that there is a blocking obstacle in Place B-C.

After the report arrives the Logistics agent (manger), the Logistics agent starts re-organization process. First process is the modification of logistics task eligibility requirement to carry package over the detected blocking obstacle. The modification is to add "*flyTo*" attribute in eligibility requirement clause. Based on the proposal, Logistics agent and other agents are exchanging messages (task announcement, bidding, and awarding) to decide next agent to handle logistics task in Place B-C area.

UAV1 agent in Place B-C has matching eligibility to handle modified task, and the task is awarded to UAV1 agent instead of the failed UGV2 agent. The task is awarded to UAV1 agent, and as the awarding decision is finalized in Logistics agent, the task termination message is sent to UGV2 agent. We have confirmed that UAV1 agent starts working on the logistics task to pick up, handover package with one another to participate the autonomous logistics task.

It is confirmed that the organized AIoT logistics application detected, interpreted and dealt with the environmental change to resiliently sustain the assigned logistics task. From the result of this experiment, it is confirmed that the AIoT devices are able to re-organize themselves by the proposed AIoT re-organization scheme in order to resiliently sustain services based on arising events. Therefore, it is confirmed that our proposal realized the resilience in IoT system service provisioning, and cleared the problem of tuning and replacing components in IoT system based on various requirements.

4.4 Evaluation of architecture-level flexibility of AIoT architecture

Since IoT system deals with heterogeneous multiple devices as well as multiple users, it generally faces various

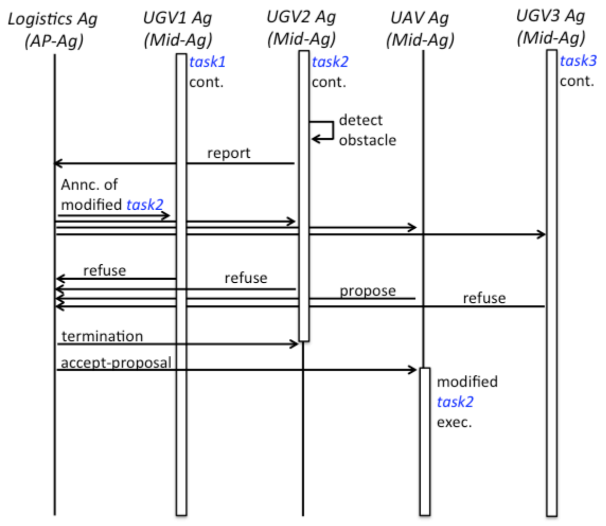


Fig. 25 Measured Message Flow of AIoT Re-organization Scheme to Sustain AIoT Logistics Application towards the Appearance of Obstacle in Task2.

changes, such as user requirement change, system condition change, and environmental condition change.

The proposed AIoT organization and re-organization capability provides autonomous construction and sustainable adaptation. These capabilities mitigate the administrative burden of IoT system, because in the conventional method, organization and re-organization performed manually. Furthermore, the AIoT architecture effectively separates the individually modifiable components as AIoT devices. It helps the modification of individual device's function in case that re-organization is not sufficient.

In order to examine the reduction of administrative cost, we have performed the architecture-level modifiability analysis (ALMA) [26]. We view the organized AIoT autonomous logistics application as an IoT system, and compare it with conventional architecture-based logistics application using the existing development middleware [7], using Node.js, REST, and HTTP technologies. The conventional system's description of components, and the system configuration are shown in Fig. 28 and Fig. 29. The component description of AIoT autonomous logistics application is shown in Fig. 30.

Since the task-oriented organization is originally proposed and provided in AIoT architecture and middleware, the conventional architecture based logistics application does not support device organization nor re-organization. Each process controlling UGV has problem solving procedures in the Node.js codes.

Table 1: Evaluation Results of Administrative Effort and Ease of Modification based on Architecture-level Comparison between AIoT Architecture based and Conventional Architecture based Autonomous Logistics Application

System Requirement Change Scenario ID	Scenario Description	Conventional Architecture based Logistics Application		AIoT Architecture based Logistics Application	
		Component Modification Ratio	LoC to modify	Component Modification Ratio	LoC to modify
Scenario 1	Replacement of single logistics robot in an area with the same logistics robot, in order to deal with the partial failure in a hardware or software.	0 out of 54 = 0 %	0 out of 5786 LoC	0 out of 17 = 0 %	0 out of 4341 LoC
Scenario 2	Replacement of single logistics robot in an area with the heterogeneous logistics robot (e.g. replacing UGV with UAV), in order to deal with the environmental change in an area.	9 out of 54 = 16 %	202 out of 5786 LoC	0 out of 17 = 0 %	0 out of 4341 LoC
Scenario 3	Addition of heterogeneous logistics robot in each area, in order to accelerate the speed of logistics.	15 out of 54 = 27 %	1012 out of 5786 LoC	0 out of 17 = 0 %	0 out of 4341 LoC
Scenario 4	Modification of logistics strategy to allow the accumulation of package in certain storage.	27 out of 54 = 50 %	1707 out of 5786 LoC	1 out of 17 = 6 %	508 out of 4341 LoC
Scenario 5	Hardware upgrade of all logistics robot to increase the speed and torque of locomotion.	15 out of 54 = 27 %	645 out of 5786 LoC	6 out of 17 = 35 %	302 out of 4341 LoC
Scenario 6	Hardware upgrade of all logistics robots to increase load and carry multiple packages at one time.	27 out of 54 = 50 %	2184 out of 5786 LoC	6 out of 17 = 35 %	990 out of 4341 LoC

Based on the ALMA method, we performed following five steps to compare AIoT architecture and conventional architecture.

Architecture-level Modifiability Analysis Steps

Step 1. *Determine evaluation goal*: We set the goal of this comparison as the evaluation of the ease of modification in case of design requirement changes.

Step 2. *Describe software architecture*: If the modification of component is necessary in IoT system, the difficulty of modification is quantifiably measured by examining the number of modified components, and the lines of codes (LoC) to be modified. We described component configuration diagram to observe the effect of system modification.

Step 3. *Elicit scenarios*: Based on the instruction in ALMA method, multiple scenarios are elicited in the logistics scenario that may require re-configuration, replacement, or modification of system components.

Step 4. *Evaluate architecture using scenarios*: In each scenario, modified (LoC) and number of components (NoC) are measured.

Step 5. *Interpret results*: To achieve the goal of comparison, the comprehensive consideration of measured results is performed.

Table 1 shows the result of the evaluation result based on the comparison between AIoT architecture-based and conventional architecture-based logistics system. We have elicited six of requirement change scenarios based on the policy of ALMA method.

In scenario 1, both systems do not need any modification since it is simply a replacement of the same component. In scenario 2, in case of AIoT architecture based system, as long as newly introduced robot is properly agentified and share PSK and CK, there is no need for the existing robots to be modified. Unlike the AIoT case, conventional architecture based system require changes in communicator parts of the existing robots to adjust the communication with the replaced robot.

Scenario 3 requires conventional system to change the communicator part, as well as the other communication related modules that manage package information. AIoT based system does not need any modification as long as the newly added device is properly agentified and share the cooperation knowledge as provided by the AIoT middleware.

Scenario 4 illustrates the effect of introducing AIoT architecture. This scenario finally requires modification in AIoT based system, however the modification is within Logistics agent (AP-Ag) to command agents in organization because AIoT architecture separates the control logic in EK and PSK in Mid-Ag. On the other hand, conventional system requires number of components to be modified.

Scenario 5 and 6 also illustrates the effect of introducing AIoT architecture. In AIoT architecture, there is no need for Logistics agent (AP-Ag) to be modified, but UGV agents and NXT agents because of AIoT architecture's layered structure of AP layer and Mid layer. In real world, UGV agent can be considered as driver agent, where NXT agent is a vehicle controlling agent. Conventional architecture requires the comprehensive modification because of ripple effect stem from the unseparated architecture.

AIoT architecture has an advantage in all of the modified LoC in each scenario. This is an effect of separating functionality of autonomous logistics application into AP-Ag and Mid-Ag which avoid the ripple effect in modification. Likewise, AIoT architecture has advantage in NoC change rate to be modified as well, except scenario 5. Scenario 5 requires 2 components' modification in each UGV, which is a relatively large number of modified components, and the conventional architecture did not cause ripple effect. Because the scenario did not require modification in components that manage packages, which results in narrower focus of modification. However, in terms of LoC, AIoT architecture has advantage in scenario 5 as well.

The results of the architecture-level comparison between AIoT architecture based logistics application, and the conventional architecture based logistics application reveals the architectural flexibility and resilience of AIoT application. It is confirmed that, in terms of design and implementation as well, AIoT architecture mitigates the administrative burden on organizing and re-organizing IoT systems, because of the effectively structured architecture of AIoT architecture.

5. Conclusion

In this paper, we proposed agent-based IoT architecture to and its middleware to realize autonomic and resilient service provisioning in IoT systems. The target problems are autonomic composition of IoT systems by taking into account the user's requirement and diverse IoT device functions, because composing IoT system involves great burden on managing heterogeneous IoT devices that require different control logics. Since IoT systems are running in physical, real world, the change of environment happens frequently. Therefore, we also targeted the tuning and replacement of IoT system components to deal with systematical, and environmental change.

To address these problems, we proposed AIoT (Agent-based IoT) architecture of application, and agentification of IoT devices, as well as the autonomic AIoT organization, and re-organization scheme to autonomously compose and operate application using the agentified IoT devices.

We have evaluated the effect of these proposals, in two of autonomous logistics experiments, as well as the architecture-level modifiability analysis. The results show that our proposed AIoT architecture and middleware address the target problems, and help developers and users realize autonomic and resilient service provisioning in the Internet of Things paradigm.

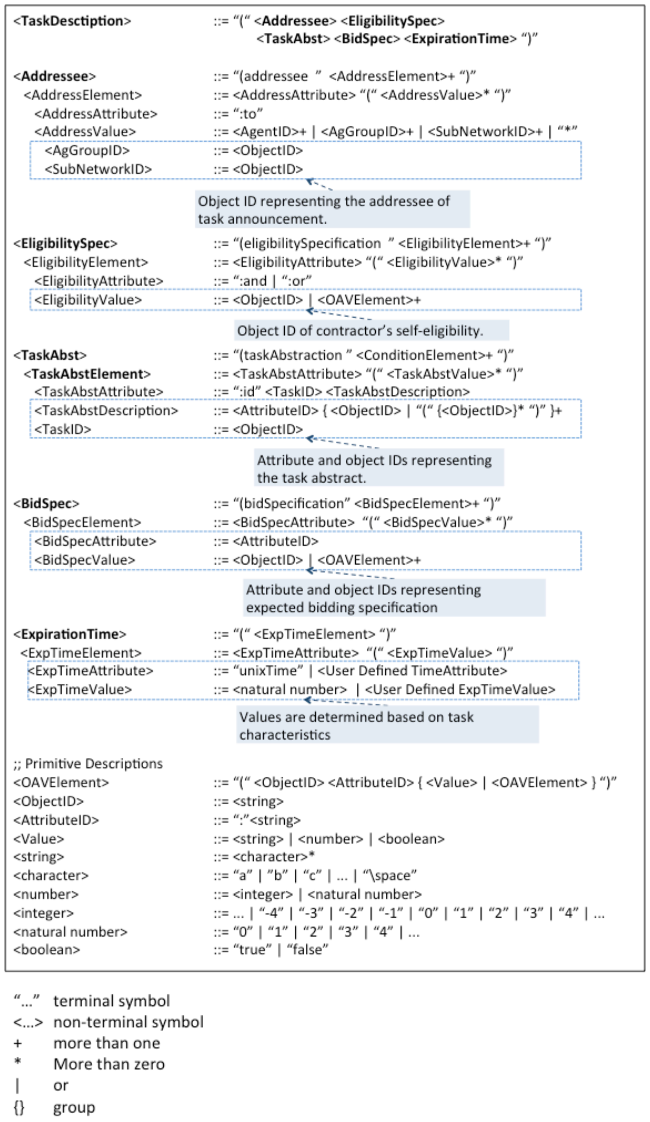
Our future work is to apply the AIoT architecture to diverse fields of application, such as home automation, energy management, smart agriculture, etc. We also plan to extend the functionality of AIoT middleware to comprehensively support AIoT application development, such as semi-automatic knowledge description generation of AIoT devices.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15J06341, 16K00118, and 16K00292.

Appendix 1 Knowledge Description Scheme of Task Knowledge and Embodiment Knowledge

Fig. 26 shows the description scheme of task knowledge for task manager in AIoT organization. The description is in OAV (Object-Attribute-Value) format. Fig. 26 is a full description of the task description in Fig. 5.

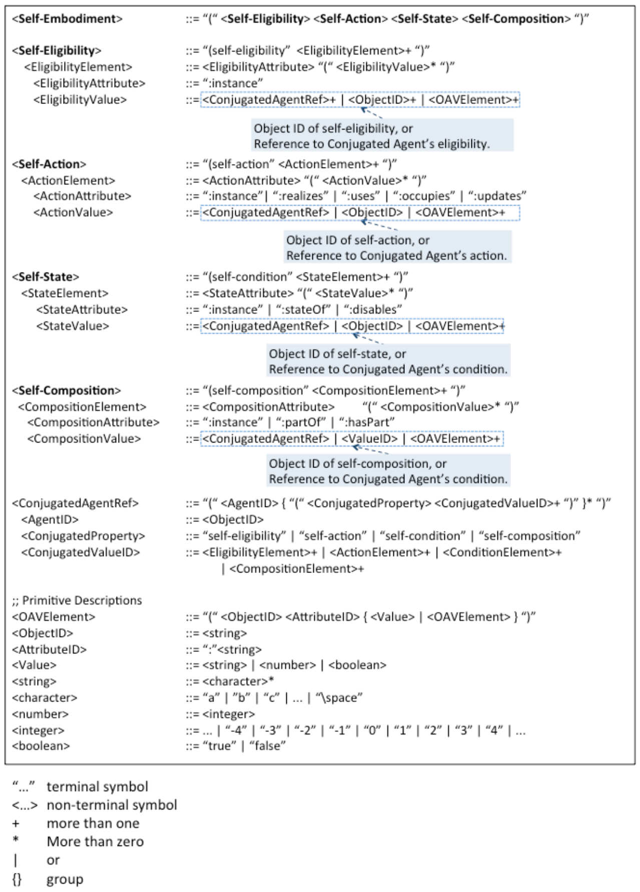


```
<TaskDesctiption>           ::= "(" <Addressee> <EligibilitySpec>
                                  <TaskAbst> <BidSpec> <ExpirationTime> ")"

<Addressee>                 ::= "(addressee "  <AddressElement>+ ")"
  <AddressElement>          ::= <AddressAttribute> "(" <AddressValue>* ")"
    <AddressAttribute>      ::= ":to"
    <AddressValue>          ::= <AgentID>+ | <AgGroupID>+ | <SubNetworkID>+ | "*"
      <AgGroupID>           ::= <ObjectID>
      <SubNetworkID>        ::= <ObjectID>

                            Object ID representing the addressee of
                            task announcement.

<EligibilitySpec>           ::= "(eligibilitySpecification " <EligibilityElement>+ ")"
  <EligibilityElement>      ::= <EligibilityAttribute> "(" <EligibilityValue>* ")"
    <EligibilityAttribute>  ::= ":and | ":or"
    <EligibilityValue>      ::= <ObjectID> | <OAVElement>+

                                 Object ID of contractor's self-eligibility.

<TaskAbst>                  ::= "(taskAbstraction " <ConditionElement>+ ")"
  <TaskAbstElement>         ::= <TaskAbstAttribute> "(" <TaskAbstValue>* ")"
    <TaskAbstAttribute>     ::= ":id" <TaskID> <TaskAbstDescription>
    <TaskAbstDescription>   ::= <AttributeID> { <ObjectID> | "(" {<ObjectID>}* ")" }+
    <TaskID>                ::= <ObjectID>

                                  Attribute and object IDs representing
                                  the task abstract.

<BidSpec>                   ::= "(bidSpecification" <BidSpecElement>+ ")"
  <BidSpecElement>          ::= <BidSpecAttribute>  "(" <BidSpecValue>* ")"
    <BidSpecAttribute>      ::= <AttributeID>
    <BidSpecValue>          ::= <ObjectID> | <OAVElement>+

                                 Attribute and object IDs representing
                                 expected bidding specification

<ExpirationTime>            ::= "(" <ExpTimeElement> ")"
  <ExpTimeElement>          ::= <ExpTimeAttribute>  "(" <ExpTimeValue> ")"
   <ExpTimeAttribute>       ::= "unixTime" | <User Defined TimeAttribute>
   <ExpTimeValue>           ::= <natural number>  | <User Defined ExpTimeValue>

                                Values are determined based on task
                                characteristics

;; Primitive Descriptions
<OAVElement>                ::= "(" <ObjectID> <AttributeID> { <Value> | <OAVElement> } ")"
<ObjectID>                  ::= <string>
<AttributeID>               ::= ":"<string>
<Value>                     ::= <string> | <number> | <boolean>
<string>                    ::= <character>*
<character>                 ::= "a" | "b" | "c" | ... | "\space"
<number>                    ::= <integer> | <natural number>
<integer>                   ::= ... | "-4" | "-3" | "-2" | "-1" | "0" | "1" | "2" | "3" | "4" | ...
<natural number>            ::= "0" | "1" | "2" | "3" | "4" | ...
<boolean>                   ::= "true" | "false"
```

```
"..."  terminal symbol
<...>  non-terminal symbol
+      more than one
*      More than zero
|      or
{}     group
```

Fig. 26 Description scheme of task knowledge for task manager in AIoT organization.

Fig. 27 shows the description scheme of embodiment knowledge for Mid-Ags in AIoT organization scheme. The description is in OAV (Object-Attribute-Value) format. Fig. 27 is a full description of the task description in Fig. 9.



```
<Self-Embodiment>           ::= "(" <Self-Eligibility> <Self-Action> <Self-State> <Self-Composition> ")"

<Self-Eligibility>          ::= "(self-eligibility"  <EligibilityElement>+ ")"
  <EligibilityElement>      ::= <EligibilityAttribute> "(" <EligibilityValue>* ")"
    <EligibilityAttribute>  ::= ":instance"
    <EligibilityValue>      ::= <ConjugatedAgentRef>+ | <ObjectID>+ | <OAVElement>+

                                Object ID of self-eligibility, or
                                Reference to Conjugated Agent's eligibility.

<Self-Action>               ::= "(self-action" <ActionElement>+ ")"
  <ActionElement>           ::= <ActionAttribute> "(" <ActionValue>* ")"
    <ActionAttribute>       ::= ":instance"| ":realizes" | ":uses" | ":occupies" | ":updates"
    <ActionValue>           ::= <ConjugatedAgentRef> | <ObjectID> | <OAVElement>+

                                   Object ID of self-action, or
                                   Reference to Conjugated Agent's action.

<Self-State>                ::= "(self-condition" <StateElement>+ ")"
  <StateElement>            ::= <StateAttribute> "(" <StateValue>* ")"
    <StateAttribute>        ::= ":instance" | ":stateOf" | ":disables"
    <StateValue>            ::= <ConjugatedAgentRef> | <ObjectID> | <OAVElement>+

                                 Object ID of self-state, or
                                 Reference to Conjugated Agent's condition.

<Self-Composition>          ::= "(self-composition" <CompositionElement>+ ")"
  <CompositionElement>      ::= <CompositionAttribute>      "(" <CompositionValue>* ")"
    <CompositionAttribute>  ::= ":instance" | ":partOf" | ":hasPart"
    <CompositionValue>      ::= <ConjugatedAgentRef> | <ValueID> | <OAVElement>+

                                 Object ID of self-composition, or
                                 Reference to Conjugated Agent's condition.

<ConjugatedAgentRef>        ::= "(" <AgentID> { "(" <ConjugatedProperty> <ConjugatedValueID>+ ")" }* ")"
  <AgentID>                 ::= <ObjectID>
  <ConjugatedProperty>      ::= "self-eligibility" | "self-action" | "self-condition" | "self-composition"
  <ConjugatedValueID>       ::= <EligibilityElement>+ | <ActionElement>+ | <ConditionElement>+
                                    | <CompositionElement>+

;; Primitive Descriptions
<OAVElement>                ::= "(" <ObjectID> <AttributeID> { <Value> | <OAVElement> } ")"
<ObjectID>                  ::= <string>
<AttributeID>               ::= ":"<string>
<Value>                     ::= <string> | <number> | <boolean>
<string>                    ::= <character>*
<character>                 ::= "a" | "b" | "c" | ... | "\space"
<number>                    ::= <integer>
<integer>                   ::= ... | "-4" | "-3" | "-2" | "-1" | "0" | "1" | "2" | "3" | "4" | ...
<boolean>                   ::= "true" | "false"
```

```
"..."  terminal symbol
<...>  non-terminal symbol
+      more than one
*      More than zero
|      or
{}     group
```

Fig. 27 Description scheme of embodiment knowledge in Mid-Ag for systematical change of its AIoT device status.

Appendix 2 Component Descriptions of AIoT Architecture-based Autonomous Logistics Application and Conventional Architecture-based Logistics Application

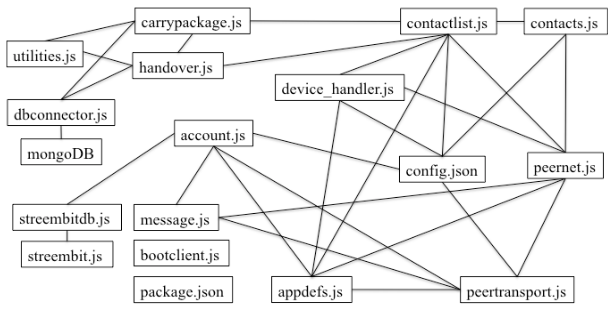


Fig. 28. Component description of conventional (Web of Things) architecture-based process to control a UGV evaluated in Section 4.4.

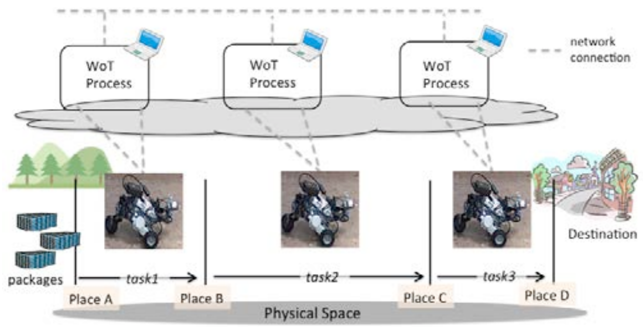


Fig. 29. System configuration of conventional architecture-based logistics application.

In autonomous logistics experiment in Section 4, we have performed a comparison between proposed AIoT architecture based logistics application, and conventional architecture based logistics application. Fig. 28 shows the component description of WoT (Web of Things) process controlling a UGV in logistics application. Each rectangle represents individual component and corresponding file names. The lines represent the references among components in code. The system configuration of conventional architecture based logistics application is shown in Fig. 29. On each UGV, WoT process in Fig. 28 is running. These processes communicate via TCP/IP network using web-based protocols.

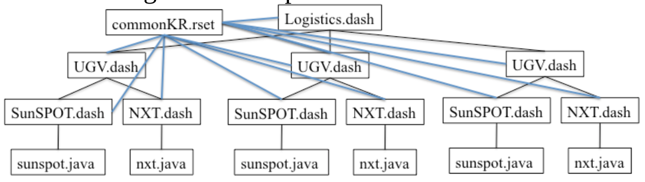


Fig. 30. Component description of AIoT architecture based autonomous logistics application evaluated in Section 4.4.

Fig. 30 shows the component description of AIoT architecture based autonomous logistics application. The files with the name ending with *.dash* are control knowledge description of ADIPS agents. The *commonKR.rset* file has sets of common rules and vocabularies among the agents. The files with the name ending with *.java* are control module processes.

References

[1] J. A. Stankovic, "Research Directions for the Internet of Things," IEEE Internet of Things Journal, vol.1, no.1, pp.3-9, Feb. 2014.

[2] A. H. Ngu, M. Gutierrez, V. Metsis, Surya Nepal, Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," IEEE Internet of Things Journal, vol.4, no.1, pp.1-20, Feb. 2017.

[3] P. Rashidi, A. Mihailidis, "A Survey on Ambient-Assisted Living Tools for Older Adults," IEEE Journal of Biomedical and Health Informatics, vol.17, no.3, May 2013.

[4] T. Tomita, K. Ushiki, Y. Kawakatsu, N. Fujino, H. Mineno, "Task-Driven Device Ensemble System Supporting Seamless Execution of User Tasks Despite Multiplexed Interruptions," International Journal of Informatics Society, vol.5, pp.49–58, 2013.

[5] L. Mainetti, V. Mighali, L. Patrono, "A Software Architecture Enabling the Web of Things," IEEE Internet of Things Journal, vol.2, no.6, pp.445-454, Dec. 2015.

[6] Node-RED, A Visual Tool for Wiring the Internet of Things. (2015.) http://nodered.org (accessed 2017.6.1).

[7] W3C Web of Things Interest Group, "Web of Things Framework for NodeJS," https://github.com/w3c/web-of-things-framework (accessed 2017.6.1).

[8] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, "Middleware for Internet of Things: A Survey," IEEE Internet of Things Journal, vol.3, no.1, pp.70-95, Feb. 2016.

[9] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, W. Russo, "An Agent-Based Middleware for Cooperating Smart Objects," in Highlights on Practical Applications of Agents and Multi-Agent Systems, J. M. Corchado Eds. Springer Berlin Heidelberg, vol.365, pp.387-398, May 2013.

[10] M. Ruta, F. Scioscia, E.D. Sciascio, G. Loseto, "Semantic-Based Enhancement of ISO/IEC 14543-3 EIB/KNX Standard for Building Automation," IEEE Trans. on Industrial Informatics, vol.7, pp.731-739, Sep. 2011.

[11] M. Ruta, F. Scioscia, G. Loseto, E.D. Sciascio, "Semantic-Based Resource Discovery and Orchestration in Home and Building Automation: A Multi-Agent Approach," IEEE Trans. on Industrial Informatics, vol.10, pp.730-741, Feb. 2014.

[12] N. Michal, K. Artem, K. Oleksiy, N. Sergiy, S. Michal, and T. Vagan, "Challenges of middleware for the Internet of Things," in Automation Control Theory and Practice, A. D. Rodić Eds. InTech, Croatia, pp.247-270, 2009.

[13] V. Terziyan, O. Kaykova, D. Zhovtobryukh, "UbiRoad: Semantic Middleware for Context-aware Smart Road Environments," in the Proc. 5th International Conference on Internet Web Applications and Services (ICIW), pp.295-302, May 2010.

[14] A. Katasonov, V. Terziyan, "Smart Resource Platform and Semantic Agent Programming Language (S-APL)," in the Proc. of 5th German Conference on Multiagent System Technologies, LNAI 4687, pp.25-36, Aug. 2007.

[15] T. Liu, M. Martonosi, "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems," in Proc. of ACM SIGPLAN symposium on Principles and practice of parallel programming, vol.38, no.10, pp.107-118, Oct. 2003.

[16] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, L. Iftode, "Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems," Computer Journal, vol.47, pp.475-494, 2004.

[17] Y. Kwon, S. Sundresh, K. Mechitov, G. Agha, "Actornet: An Actor Platform for Wireless Sensor Networks," in the Proc. of Fifth International Joint conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), pp.1297-1300, May, 2006.

[18] C.-L. Fok, G.-C. Roman, C. Lu, "Agilla: A Mobile Agent Middleware for Self-adaptive Wireless Sensor Networks," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol.4, no.3, pp.1-26, July 2009.

[19] R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Trans. on Computers, vol. C-29, pp.1104-1113, Dec. 1980.
[20] A. Schuldt, "Autonomous Control in Logistics," in Multiagent Coordination Enabling Autonomous Logistics, A. Schuldt Eds. Springer-Verlag, Berlin Heidelberg, pp.37-69, 2011.
[21] IDEA: Interactive Design Environment for Agent System, http://www.k.riec.tohoku.ac.jp/s/idea/ (accessed 2017.6.1).
[22] S. Fujita, H. Hara, K. Sugawara, T. Kinoshita, N. Shiratori, "Agent-Based Design Model of Adaptive Distributed Systems," Applied Intelligence, vol.9, pp.57–70, 1998.
[23] SunSPOT, http://www.sunspotdev.org/ (accessed 2017.6.1).
[24] LEGO Mindstorms, http://www.lego.com/en-us/mindstorms/ (accessed 2017.6.1).
[25] Parrot AR. Drone 2.0, http://global.parrot.com/jp/products/ardrone-2/ (accessed 2017.6.1).
[26] P. Bengtsson, N. Lassing, J. Bosch, H. Vliet, "Architecture-level modifiability analysis (ALMA)," The Journal of Systems and Software, vol.69, pp.129–147, Jan. 2004.

Takumi Kato is currently a Ph.D. candidate at the Graduate School of Information Sciences (GSIS), Tohoku University, Japan. He received his B.S. in Engineering from Sendai National College of Technology in 2011, and the M.S. in Information Sciences from Tohoku University in 2013. His research interests include Internet of Things (IoT), knowledge-based system and agent-based computing. He is a member of IPSJ and IEEE.

Hideyuki Takahashi is an assistant professor of Research Institute of Electrical Communication of Tohoku University, Japan. He received his doctoral degree in Information Sciences from Tohoku University in 2008. His research interests include ubiquitous computing, green computing and agent-based computing. He is a member of IEICE and IPSJ.

Tetsuo Kinoshita is a professor of Research Institute of Electrical Communication of Tohoku University. He received the B.E. degree in electronic engineering from Ibaraki University, Japan, in 1977, and the M.E. and Dr. Eng. degrees in information engineering from Tohoku University, Japan, in 1979 and 1993, respectively. His research interests include agent engineering, knowledge engineering, knowledge-based systems and agent-based systems. Dr. Kinoshita is a member of IEEE, ACM, AAAI, IEICE, IPSJ, and JSAI.