Architecture and framework for data acquisition in cloud robotics

Y. Watanobe*, Y. Yaguchi, K. Nakamura,
T. Miyaji, R. Yamada and K. Naruse

Department of Computer Science and Engineering,
University of Aizu,
Aizuwakamatsu, Japan
Email: yutaka@u-aizu.ac.jp
Email: yaguchi@u-aizu.ac.jp
Email: keita-n@u-aizu.ac.jp
Email: m5211146@u-aizu.ac.jp
Email: ryamada@u-aizu.ac.jp
Email: naruse@u-aizu.ac.jp
*Corresponding author

Abstract: This study explores a data acquisition architecture and framework in cloud robotics. In cloud robotics environments, software components play important roles by acquiring data from heterogeneous devices and then performing context-aware computing with the help of knowledge bases organised by the data. However, there are numerous tasks that must be performed to create such components, and their corresponding database schemas, services, and repositories, and these tasks can be burdensome for developers. In this paper, an architecture and framework for constructing a data acquisition system are presented as a theory and a concrete implementation, respectively. Our proposed architecture enables developers to construct a robot environment with data acquisition functionalities by defining scenarios in an ontology language, as well as by defining objects and services in modern programming languages. The framework is realised in a way that allows it to automatically generate the required software components and their corresponding repositories, which are deployed on the cloud. Case studies showcasing the proposed framework are also presented.

Keywords: cloud robotics; architecture; data acquisition; robotics technology components.

Reference to this paper should be made as follows: Watanobe, Y., Yaguchi, Y., Nakamura, K., Miyaji, T., Yamada, R. and Naruse, K. (2021) 'Architecture and framework for data acquisition in cloud robotics', *Int. J. Information Technology, Communications and Convergence*, Vol. 4, No. 1, pp.1–25.

Biographical notes: Y. Watanobe is currently a Senior Associate Professor with the School of Computer Science and Engineering, The University of Aizu, Japan. He received his MS and PhD from the University of Aizu in 2004 and 2007 respectively. He was a Research Fellow of the Japan

Society for the Promotion of Science at the University of Aizu in 2007. His research interests include data mining, smart learning, cloud robotics, and visual languages.

Y. Yaguchi received his BS in Computer Science and Engineering from the University of Aizu in 2006, and his PhD in Computer Science and Engineering from the Graduate School of Computer Science and Engineering, University of Aizu in 2011. In 2011, he became an Assistant Professor in the Department of Information Systems, Faculty of Computer Science and Engineering, University of Aizu, and in October 2013, he became an Associate Professor in the same department. His research interests include navigation and path planning of multiple UAVs, security and safety co-engineering IoRT and cloud robotics design.

K. Nakamura received PhD in Information Science from Hokkaido University, Japan in 2013. He worked as an Assistant Professor at the Department of Information and Computer Engineering, National Institute of Technology, Gunma College, Japan from 2014 to 2016. He is currently an Associate Professor at the University of Aizu, Japan. His research interests include robot performance evaluation method, 3D object reconstruction, deep learning, robot vision, data mining, natural language processing.

T. Miyaji is a master student of University of Aizu. His research interests include cloud robotics and database management systems.

R. Yamada is an Associate professor at the University of Aizu. He received his PhD of Earth and Planetary Science at the University of Tokyo. He has research experiences in University of Toulouse III, Japan Aerospace Exploration Agency and National Astronomical Observatory of Japan. His major studies are development of sensors for planetary exploration, geophysics, planetary science and robot engineering such as autonomous moving robots.

K. Naruse is currently a Full Professor with the School of Computer Science and Engineering, The University of Aizu, Japan. His research interests include swarm robots and its application to agricultural robotic systems, and interface system for disaster response robots. Currently, he has worked for design, development, and standardisation of networked distributed intelligent robot systems with heterogeneous robots and sensors. He has applied it to service robot systems, factory automation systems, and intelligent disaster respond robot system, which is tested at Fukushima Robot Test Field.

This paper is a revised and expanded version of a paper entitled ‘Data acquisition framework for cloud robotics’ presented at 2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST), Morioka, Japan, 23–25 October 2019.

1 Introduction

Currently, cloud technology is one of the most trending buzzwords in information and communications technology (ICT), which provides new opportunities for different application domains. In the area of robotics, cloud technologies can be used to realise an approach that offloads heavy calculation and memorisation tasks away from the robots by using cloud computing as well as edge computing. For example, a variety of architectures and challenges have been explored by Chen et al. (2018a), Saha and Dasgupta (2018) and Chen et al. (2018b).

One of the attractive features of cloud robotics, or the internet of robots (IoR), is that robots can access data from a vast knowledge base that was initially constructed by other robots in order to facilitate context-aware computing. That knowledge base can also be utilised for different purposes, including both real-time and batch processing. Furthermore, current trends of big-data analysis and frameworks for artificial intelligence (AI) are also accelerating innovation in the cloud robotics field (for example see Guo et al., 2020; Wang et al., 2020).

Meanwhile, recent technological advancements in web-space environments and their corresponding databases are providing fertile ground for continued growth in cloud computing. This trend is most easily seen in the ever-increasing number of web engine frameworks, database management systems, distributed systems, knowledge representation techniques, and virtualisation technologies that are now available to efficiently implement and deploy cloud-accessible web services through HTTP. Those services are also capable of developing and operating in a big-data analysis environment for different application domains. For example, representational state transfer (REST) APIs or RESTful APIs provided by microservices can provide an architectural basis for both the internet of things (IoT) and document management system developments, as well as for hardware integration (for examples see Al-Masri, 2018; Williams et al., 2014; Ezzeddine et al., 2018).

Although several previous studies have focused on architecture, in terms of the software development strategies for robot systems, middleware and operating systems are key technologies that provide the communications interfaces needed for integrating heterogeneous components. In the field of cloud robotics, software components that can acquire/obtain data from different sources/repositories for knowledge sharing will take on increasingly important roles, while ordinal functions need to be found for legacy components related to hardware devices, controllers, and algorithms that can be reused as software assets. In this context, the active development of robot systems will depend on the integration and interoperability of legacy and advanced data acquisition components. Furthermore, while many current projects aim at integrating robot systems and web platforms, the flexibility, interoperability, and availability of existing software assets will depend on technologies employed in target projects.

In this paper, the architecture and framework of a data acquisition system for cloud robotics are presented. In our project, the cloud robotics are oriented toward environments in which different types of robots, such as indoor and outdoor multi-service robots, etc. collaborate by sharing data with each other on disaster scenes, etc. More specifically, the framework is oriented towards constructing a cloud robotics environment based on robotic technology components (RTCs), in which data elements obtained from the controllers, actuators, and sensors of various robots are collected into a robot data repository (RDR). This allows the data to be used for different purposes, both offline and online, in order to make the robot system more intelligent. Generally speaking, since the raw data collected from heterogeneous robots is disorganised, it needs to be transformed into well-organised data to facilitate effective search operations. These transformation processes are performed by different types of manipulations through multistage databases. However, to construct such robot systems, various burdensome processes must be completed. These include the development of components that are compatible with employed technologies, the design of database schemas, the implementation of APIs, as well as the deployment of necessary controllers, services, and repositories on the server side. Another point of concern is that, depending on scenarios, the required components and the corresponding schema may vary, even within the same project. As a result, developers may need to construct systems for each scenario or experiment.

With these points in mind, this paper presents an architecture and framework that supports the developers by providing a unique builder that automatically generates reusable software components for data acquisition, database schemas, and their corresponding repositories. It also provides services that can be used to transform raw accumulated data into usable forms within the multistage databases. The automated processes are performed within five layers oriented toward ontology, programs, software components, web services, and database repositories. Hence, the contribution of this paper is summarised in the following three areas:

- We propose an architecture that provides a theoretical basis for the automation processes used for system integration within robotics environments.
- We show how the framework is employed based on the proposed architecture as a concrete implementation with modern technologies.
- We present two case studies that show how the framework can seamlessly create the required components, services, and repositories.

Herein, two case studies are presented. The first study relates to simple data transformation based on definition through ontology, while the second relates to a scenario in which different robots cooperate through data shared on the cloud. Through these case studies, the productivity, availability, sustainability, scalability, and transparency of the proposed architecture and framework are demonstrated. This paper is an extension of Watanobe et al. (2019) originally presented in *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)* titled 'Data acquisition framework for cloud robotics'.

The rest of the present paper is organised as follows. In Section 2, related work is presented. In Section 3, overviews of the cloud robotics concept and data acquisition in the cloud robotics field are presented. In Section 4, the general scheme of the architecture and its concrete framework implementation are discussed. The case studies and their related discussion are presented in Sections 5 and 6, respectively. Finally, we conclude the paper in Section 7.

2 Related work

There have been a wide variety of studies aimed at making robots more intelligent by means of machine learning or other data analysis technologies (see for example Popov et al., 2018; Bergamini et al., 2020). However, even though a number of case studies have been demonstrated previously, our study focus is on how we can efficiently develop a system in which the robot's brain is maintained on the cloud along with the required data. Within that context, there have been a number of different studies that have some relationships with our research activities, including cloud robotics, data repositories, architecture, and standardisation.

Many projects aim at providing a platform as a service (PaaS) for cloud robotics. The first we should mention is the RoboEarth project by Waibel et al. (2011), in which the RoboEarth cloud engine and RoboEarth database are integrated to construct a World Wide Web (WWW) for robots. There is also Rapyuta, initiated by Hunziker et al. (2013), which is the name of a cloud engine that allows developers to deploy a safety server based on Linux containers. Using this platform, robots can access the server to offload computations and submit queries to the knowledge base through nodes uploaded by the developer. The project presented by Haidu et al. (2018) is focused on the engines for reasoning, prediction, and learning tasks that are required for object manipulation.

Similar to the projects mentioned above, OpenEASE, initiated by Bozcuoglu et al. (2018) and Beetz et al. (2015) is a web-based knowledge service that consists of a big-data database of manipulation information obtained from both humans and robots. It includes the manipulation of concept models, as well as software tools for querying, visualising, and analysing those manipulations.

As for data repositories, there have been various proposals related to cloud robotics architectures. For example, Niemueller et al. (2012) proposed a system based on the MongoDB database and the robot operating system (ROS) for fault analysis and performance evaluation applications. In addition, Park et al. (2016) proposed a system architecture that could be used to control robot tasks through sensory data acquisition in ROS environments, while Filip et al. (2019) proposed an architecture for a cloud-edge infrastructure, along with a model and a formal description of a data capsule.

Although the projects mentioned above are oriented on the ROS system, there are several ongoing projects for developing robot systems with standardisation. These include RT-middleware (RTM) initiated by Ando et al. (2011), which is one of the promising projects based on the object management group (OMG) standard, and a number of robotics technology components that have been developed as software assets, including intelligent modules (for examples see OpenRTM-aist, https://www.openrtm.org/openrtm/; RTC-Library-FUKUSHIMA, https://rtc-fukushima.jp/; Yaguchi et al., 2017; Ogitsu et al., 2015; Tsuichihara et al., 2015).

On the other hand, to integrate costly reusable legacy components from different projects and expand them as Internet services, gateways between RTM and common network-based robot service platforms have been considered by Kato et al. (2011) and Narita et al. (2009). Hence, the continuous development of OpenRTM (https://github.com/OpenRTM) with different communication interfaces, which takes interoperability (or bridging) with ROS into account, is also an attractive feature of our project. For example, brokered publish-subscribe (pub/sub) messaging interfaces for RTM have been considered by Yoshino et al. (2017) and communication interfaces based on the advanced message queuing protocol (AMQP) and the mosquitto MQ telemetry transport (MQTT) message broker, both of which are compatible with RTM, have also been studied by Yoshino et al. (2019) and Yoshino et al. (2018) for IoR systems.

Finally, we should discuss ways for representing standardised knowledge, which can facilitate human specification of different experiences as well as automated machine processes. In this context, a standard ontology that specifies entities related to robots and robotics environments, as well as their relationships, has been defined by IEEE Standard Ontologies for Robotics and Automation (2015). Ontology is a useful technology for defining information in a standard way as well as in a machine-readable format, which is why it has been employed in various domains including software engineering, robotics, and other data science fields. For example, ontologies for robotics and autonomous system have been considered by Fiorini et al. (2017) for task definition, automation, and other communication entities between humans and robots. Another example is ontology building for cyber-physical systems (CPS), which was proposed by Hildebrandt et al. (2020). In contrast to those efforts, we are trying to employ ontologies in the automation process in order to generate data acquisition components.

Although, as mentioned above, several potential projects provide AI engines at higher levels, it is difficult to find a framework that generates interoperable components that are compatible with specific projects and which are equipped with corresponding repositories that are designed for specific scenarios. Furthermore, our focus is also on multistage database repositories for low-level data manipulation, including transformation, data cleaning, interpolation, and the like. The utilisation of modern technologies for web and database management systems that are easily deployed on dedicated host servers is also one of the unique points of our approach. In this context, it is also in our interest to explore software architectures and design patterns. For example, our proposed architecture and its implementation are influenced by the dependency injection (DI) idea proposed by Inversion of Control Containers and the Dependency Injection Pattern (https://www.martinfowler.com/articles/injection.html), where a dependency can be inserted as a service into an object that will use it.

3 Cloud robotics concept

In this section, a brief overview of the cloud robotics we pursue is presented. Cloud robotics is based on the concept of creating systems where robots, sensors, humans, and other things are organically connected within a number of different clients and servers. Figure 1 shows our cloud robotics concept from different perspectives, including how we should develop and deploy a robot system by employing legacy components from the library. The robot system can be deployed in an actual environment or in cyberspace. The figure also shows how to make robots more intelligent by using data repositories with AI algorithms. Additionally, some RDR features are presented through data characteristics, the structure of the repositories, as well as development and deployment processes.

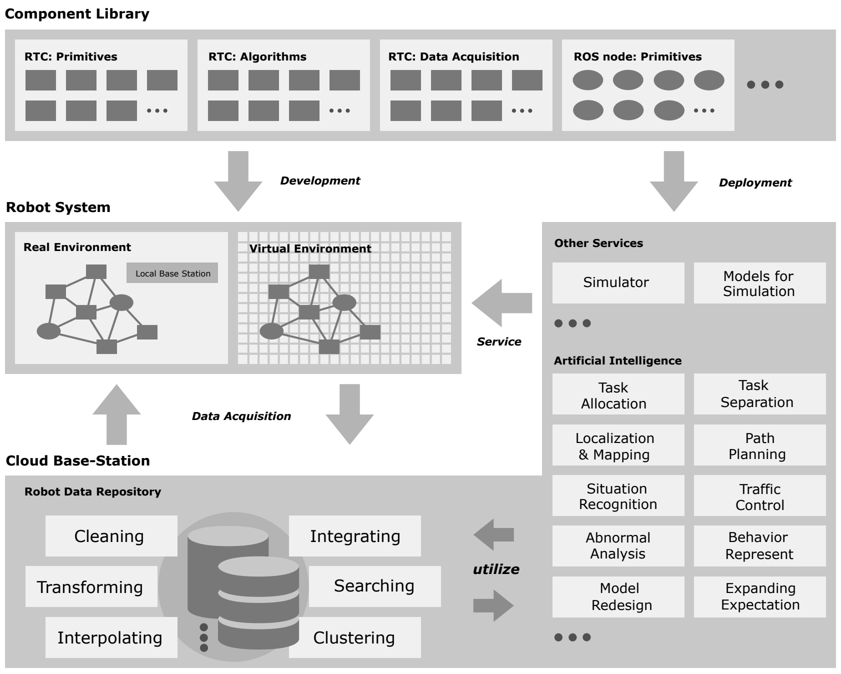
3.1 Robot system

Although there are a number of different robot types, as defined by ROBOTS (https://robots.ieee.org/learn/types-of-robots/), our project employs the following robots for use in different scenarios.

- *Large double-arm robots* that are designed to clear away large chunks of debris. In a disaster, robots will be remotely controlled from the base station. Therefore, video images and other information such as circumjacent three-dimensional (3D) point data need to be sent to the stations, which will use them to make decisions related to the robot's arm movements and its path of travel.

- *Medium-size crawler robots* that prowl around and through debris on the ground. These crawler robots can travel through rutted and sloped areas that large robots are unable to traverse while simultaneously recording the actual conditions in the form of images and video footage from different directions. Such imagery is useful for appraising the genuine conditions around the robots, helping to locate survivors, and identifying danger points in the wreckage. Generally speaking, the crawler robot is flexible and can be easily expanded by mounting additional devices such as arms and microscopes.
- *Small crawler robots* that can explore more flexibly on the ground. In addition, cameras, different sensors, and devices such as sound recognitions systems can be mounted to these small robots. One of the crucial roles of such small robots is to facilitate data transfers by serving as mobile relay stations on the ground.
- *Unmanned aerial vehicles*, commonly called drones, which can look down and observe the ground continuously. While their primary role is to send bird's-eye imagery data and spatial information to the base stations and other robots, environmental information is also recorded. It is also important to remember that several drones can be organised in a cluster and fly in formation in order to collect data covering a wide area over a long period. Such drones can also serve as mobile stations to facilitate data transfers between different robots.

Figure 1 Overview of cloud robotics



Each robot type consists of motors, actuators, and related sensors, and each robot's movements are based on the accumulation of its simple device behaviours. This means that the robots can improve their intelligence by analysing their own log data.

Furthermore, the various types of data show that the robot system as a whole can be made more intelligent by sharing data and by collaborating in ways that allow each robot to fulfill its unique functions. This cooperation is a crucial issue for disaster scenes as can be seen in the challenges of the WRS (https://worldrobotsummit.org/en/) in which such situations are simulated with the abovementioned robots.

Local base stations are setup to provide remote control for different robots based on both human and machine decision-making processes. Local base stations also provide environments where images, robot statuses, and other environmental information can be observed on display screens. Robot controllers and user interfaces are also deployed at such stations. Such local stations can also be connected to cloud base stations in order to obtain relevant data and optimal instructions.

Simulators, where operators can manipulate robots or check on their behaviour via a virtual environment, are also essential parts of robot systems for practice and related experiments, while the data and related services provided by the cloud play important roles in related simulations.

3.2 Cloud base station

The cloud base station is the brain of the robot system and is oriented to both real-time processing (by online accumulation) and the organisation of an optimal environment (by offline accumulation). The intelligence, safety, and reliability of the robot system are dependent on both the quality and quantity of data. Therefore, useful information resources from available sources should be thoroughly accumulated, and the data should be refined through multistage databases.

At the lower level, the raw data should be cleaned via noise removal, interpolation, integration, and other transformations through the multistage databases. To manage the data, database management systems, appropriate storage areas, and the corresponding application programming interfaces (APIs) should be deployed to bridge the gap between the stations and the robot system (and other services).

In contrast, a variety of AI engines should be deployed at a higher level. These engines utilise the data in the repositories and provide added value to the robot system, as well as to the repositories. As for other services, simulator platforms with related models are among the critical parts of a cloud base station (note that details regarding AI engines and simulators are considered to be outside the scope of this paper).

3.3 Component library

In terms of the development side and the robot system deployment, reusable legacy components with a common interface play a key role in integrating heterogeneous functional elements, which include controllers for different hardware devices (such as motors), sensors for obtaining different types of data (such as cameras), and algorithms for various AI functions (such as image processing). Since such modules can be prepared from reusable legacy software components, the development and popularisation of a library for such components have the potential to facilitate high-quality and rapid development, as well as enhance cost reductions via recycling. To communicate with the cloud base station, software components for data acquisition (as well as AI engines) should be implemented as members of the component library. In other words, the legacy components and the data acquisition components should coexist so that their reusability and interoperability are maintained. Regarding deployment, if a data acquisition component is involved in the robot system, the corresponding repository and API should be prepared on the cloud base station.

3.4 Data

Our first target comprises data related to the status of each robot, which are useful for organising and ensuring optimal and safe robot movements. For all robots, timestamps, positions, accelerations, velocities, and angles, as well as their foreseeable and derived values (such as distances and duration), are necessary for optimal and automated control. They are also essential data for creating safe robot environments where danger avoidance and emergency stop functions are appropriately activated.

In addition to data from robots, log data from their controllers at the local base station, as well as from the robots themselves, should also be utilised. Consistencies between actual robot movements and their corresponding command logs can be analysed to identify potential accident scenarios, as well as to improve the robot system as a whole.

In addition to information related to raw data, multimedia data from sensors (including cameras) take on essential roles in many scenarios. For example, images related to bird's-eye views, environmental data (temperature, pressure, etc.) from sensors, and other information (such as sounds) are employed to gain an overall understanding of current and past conditions. Point sets obtained by 3D LiDAR can be used for creating maps and environments based on simultaneous localisation and mapping (SLAM).

3.5 Structure of repositories

The RDR is based on the idea of a data lake in which different operations such as ingestion, extraction, cleaning, integration, and discovery are accomplished (see for example Nargesian et al., 2019). Since heterogeneous devices exist in cloud robotics environments, data will arise from different sources at different times and at various frequencies. This means that, depending on circumstances such as latency and accuracy, data elements are often collected irregularly even if they are obtained from the same type of robot. However, it is difficult to manage such data using only a single relational database in which the elements of each record are expected to be ready simultaneously based on pre-defined attributes. Therefore, as shown in Table 1, the RDR data are managed within multistage databases.

Data obtained from the sources at the very beginning should be accumulated in the first-order database, which is oriented towards saving data 'as is'. On the other hand, the data manipulations should be performed through the multistage databases (second-order databases) involving intermediate data, after which they are transferred to one of the second-order databases oriented towards searching from different perspectives.

The first-order database organises unstructured (semi-structured) data. Unstructured data are by nature raw data, and each record can be represented as an object. Since unstructured data repositories will receive data with different types and sizes at irregular intervals, they will eventually become massive and cumbersome if left in an unstructured state. To make them useful, such repositories need to be implemented via object-oriented databases with key-value stores and NoSQL technology for sorting and storing the unstructured data.

Table 1 Features of data in different stages

Robot data repository	
1st-order database	2nd-order databases
Collected as raw data	Re-organised as cleaned data
Stored and organised 'as is'	Stored after integration and combination of data from different sources
Wide range of uses as future assets	Meet specific reuse requirements
Different data types and sizes	Exist in predefined fixed fields
Collected in irregular periods and formats	Provided on demand
Massive and cumbersome data	Refined data
Unstructured, semi-structured data	Structured data
Managed by key-value	Managed by RDBMS

In contrast, since the second-order databases need to be able to provide useful information from the structured data, the repository aims to provide data according to on-demand requests quickly. Accordingly, the repository can be organised in a relational database with predefined fields for various searching operations, but the structured data schema must be well defined in terms of field names and their data types.

3.6 Processes for development and deployment

When considering robot projects, there is a wide variety of scenarios involving different data sources and sinks depending on the scenario's purpose. As a typical scenario example, a crawler robot scans a building to obtain information regarding indoor objects, after which it sends point sets and image data obtained by its LiDAR and cameras to the RDR. Then, using multistage databases, those information resources can be cleaned, integrated, and transformed into structured data (through machine learning if necessary) so that other robots in that building and other systems can utilise the results for navigation and other intelligent services. Examples of such services include creation of 3D map and object recognition (see for example Funayama et al., 2020).

To refine such robot systems, actual machine operations and experimental simulations are performed in physical environments and virtual spaces, respectively. Hence, as shown in Figure 2, developers (system integrators) need to create a number of data acquisition components for each scenario. Furthermore, the corresponding repositories and transformers should be deployed in the RDR.

Therefore, the role of software engineering is to provide a way to create the required items and reduce associated burdensome tasks which the developer needs to perform the following actions:

- design a set of database schemas for the different stages
- create transformers to reorganise data in the first-order database for use in the second-order databases (multistage databases)

- implement corresponding data acquisition components that are compatible with other components integrated into the robot system, as well as with the database
- deploy and activate controllers (bridges between the components and repositories), services (such as transformation programs), and repositories on the server.

Figure 2 Development scenario for robot system data acquisition

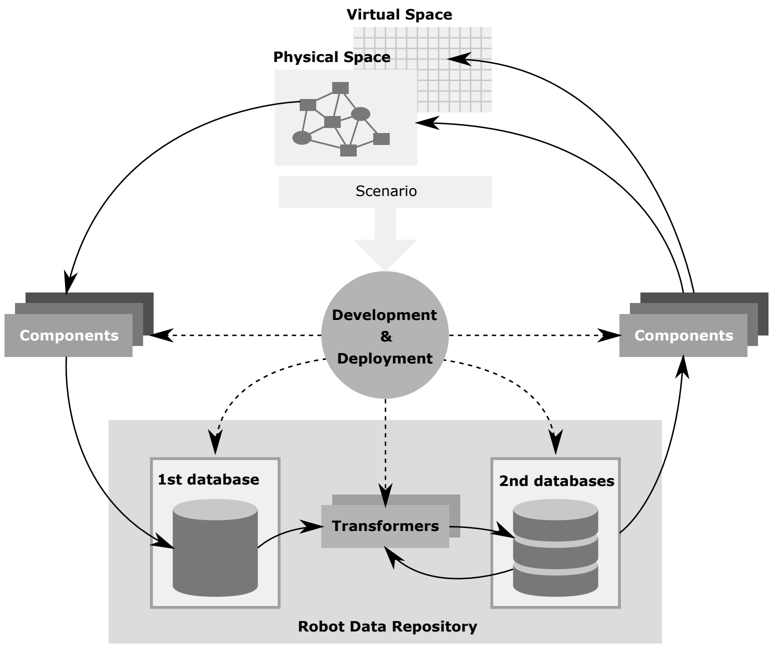
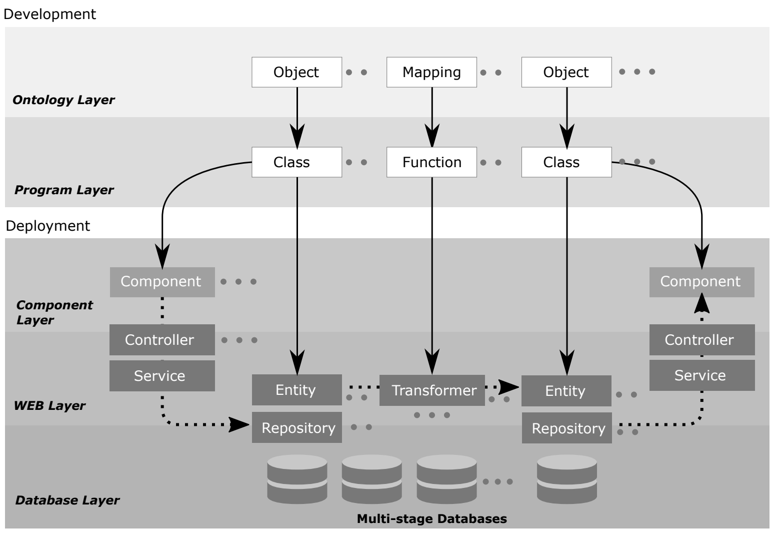


Figure 3 Overview of the proposed architecture



4 Architecture and framework

The RDR is oriented on a data-centric architecture where the developers design their robot systems by focusing on the sources, repositories, and receivers of data as well as on how they are transferred through various components. In this section, the architectures of the RDR and its compatible framework are presented.

4.1 Architecture

As mentioned above, the processes used to construct a data acquisition system include a number of tasks. However, developer requirements should be limited and processes should be automated whenever possible. Generally speaking, the developer should only need to define the scenario data that he/she needs to manage. Hence, the aim of the architecture and framework is to support developers via a method with which they can define the data involved in the robot system in a uniform style or in an accustomed manner. Then, the framework should automatically generate and deploy the necessary components, APIs, controllers, services, and repositories.

Since it would be difficult to implement this framework with a monolithic architecture, it should instead be organised on several layers, each of which is dedicated to its function and implemented via the appropriate technology. Figure 3 shows an overview of our proposed architecture, which realises the automation processes through five layers:

1 an ontology layer

2 a program layer

3 a component layer

4 a web layer

5 a database layer.

In the figure, the continuous line arrow shows the automatic generation, and the dotted line arrow shows the dataflow during actual operation. The first top two layers are oriented toward the development perspective, where system integrator activities are considered. In contrast, the bottom three layers are oriented toward the deployment perspective, where generated components are operated at different places. Each of these layers is discussed in detail below.

The ontology layer provides a way to define data and their relationships through the highest user interface. This layer exists to support humans in different groups, including non-programmers, while presenting the knowledge specified by humans in a machine-readable format. The knowledge shared by robots in the ontology layer is defined in the OWL Web Ontology Language (https://www.w3.org/TR/owl-features/), which is a family of languages for knowledge representation. These languages, which are implemented by the World Wide Web Consortium's (W3C) XML standard for objects, allow developers to define required data through the standard without depending on their knowledge of special technologies (such as programming languages). Since the ontology can be defined in advance for the whole project, he/she selects required objects and relationships from the available items in the defined ontology for each scenario. The ontology includes a set of objects and relationships (mapping) between them.

The program layer mediates between knowledge defined in the ontology and the components, which are created in the bottom layers. It exists to provide system integrators, who will want to use programming languages to define various entities and their transformations, with opportunities to define knowledge. After a set of objects/relationships are selected for a scenario in the ontology layer, the required programs are generated. There are two program types: data and transformer. A data program is a model that is used to represent an object defined in the ontology layer. In contrast, a transformer program is a function that transforms one data object to another data object based on the corresponding mapping in the ontology (in the same manner as used in the ontology layer).

The component layer provides the main products of the framework. In this layer, based on the created programs, the components that are to be integrated into the local robot system are generated for the client side. A component is a software module that is compatible with RTCs or nodes for the corresponding robot system. There are two component types. One is the write component, which receives data from connected components and sends it to the database on the cloud. An ontology (or program) element defined in the above layers is used to define the input interface of the write component. The other is the read component, which provides data from the database based on requests from other connected components. An ontology (or program) element defined in the above layers is used to define the output interface of the read component. In principle, a component can be generated from any object defined in the ontology or program layers.

The web layer provides an interface between the components in the local system (client) and the available data on the cloud, that is, the data repositories. In other words, it exists to provide APIs. Hence, after items in ontology layer and program layer are ready, a set of services, controllers, and entities are generated in the web layer. An entity is a model that represents data, and the accompanying repository contains available operations compatible with the corresponding databases. A transformer is a special service that is periodically performed to transform records in one database into records in another database. Generally speaking, transformers are created based on elements defined in the above layers. It is important to note that transformers can be automatically created if the relationship between the entities involved is defined in the ontology as a whole. The controllers bridge the gaps between the components on the client side and the databases on the cloud side through accompanying services.

The database layer is the RDR core where all data is managed. In this layer, databases are deployed based on repositories with required schemes and tables. A database entity is created from an entity, and data from the write components are accumulated into one of the first-order databases through their corresponding repositories. In contrast, the transformers create and update records in the second-order databases that provide data to the read components through their corresponding repositories.

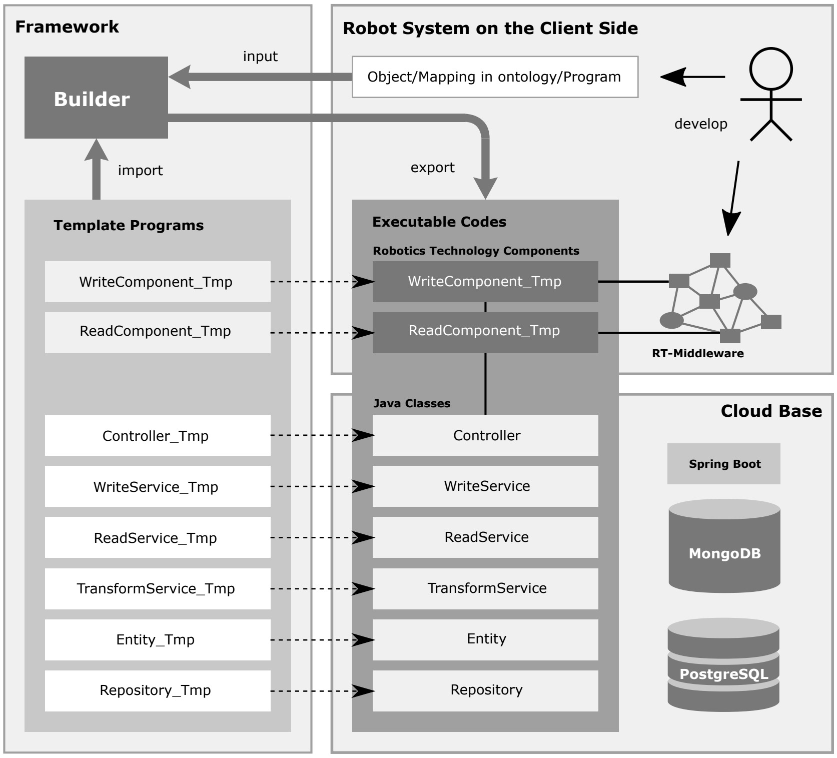
4.2 Framework

It is particularly notable that the architecture presented in the previous section provides a theoretical concept that can be implemented by integrating different technologies and existing tools. We also implemented a framework oriented to RTCs that is compatible with OpenRTM. Figure 4 shows the procedures and elements involved in the framework, which contains the primary builder used to export the required executable codes from scenario definitions or programs. The primary builder was developed in Python. The builder imports a set of template programs related to the RTC, service, entry, repository, and controller classes. Although RTCs can be created in C++, Java, or Python, we focused on generating RTCs in Python.

In the ontology layer, to prepare the ontology, developers can employ an existing ontology editor that is compatible with OWL. We employed Protege (https://protege.stanford.edu/) for our framework. This ontology editor provides a graphical user interface (GUI) that can be used to define objects as entities and their hierarchical configuration as well as their relationships. Such editors can consequently export the ontology in the XML standard format, which can then be input into the following layers.

In the program layer, the framework generates classes in Java related to data and transformers from the corresponding ontology in XML. Optionally, the developer can also create such classes from scratch by skipping the ontology definition. This option is provided to support programmers who are familiar with traditional programming languages. A data class consists of fields that represent attributes and their setters/getters. In contrast, a transformer class consists of functions that generate new objects from given objects through the required calculations.

Figure 4 Procedure and elements in the implemented framework



In the component layer, RTCs are generated based on the ontology or Data classes. RTCs for writing and reading are generated as WriteRTCs and ReadRTCs respectively. The RTC communication interfaces are compatible with the other RTCs involved in the environment. An RTC provides data ports for the corresponding data fields (attributes). A WriteRTC sends the data that has arrived at the ports to the cloud station through POST operations. In contrast, a ReadRTC obtains data from the cloud station through GET operations.

In the web layer, the framework employs Spring (https://spring.io/) to provide the RESTful API. The Spring Framework employs a software architecture pattern called the DI mechanism for services and repositories and is compatible with the proposed architecture. In this layer, classes for data and transformer are converted into the classes of entity and service, respectively. The framework provides mechanisms to generate repositories (databases) from the entity classes automatically. The framework deploys and activates the created controllers, services, and repositories on the dedicated cloud server. Furthermore, the framework can launch processes to periodically activate specific functions in a service.

In the database layer, the objects and tables of MongoDB and PostgreSQL are generated based on the repositories for the first- and second-order databases, respectively. The framework deploys these databases on the dedicated cloud server. Databases employed in the cloud station can be specified through framework properties. After activating the application, controllers reside in the cloud base station where they deal with requests from the outside. Functions in services are periodically performed for transforming data between the multistage databases.

The implemented framework supports not only the generation of necessary items but also their deployment. After the items are ready, the required controller and services are activated with the specified preferences on the cloud. On the other hand, even though, ideally, conventional querying options would be available for the RTC that is oriented to read data from the RDR, the implemented framework has limitations. The current implementation supports just a few representative options, including the timestamp-based search operation that is embedded in every element.

5 Case studies

In this section, case studies that employ the implemented framework are presented from different perspectives. Since we focus on demonstrating how the required parts of the robot system and the cloud base station are created and deployed seamlessly by the framework, we selected two simple ad-hoc applications, one of which is related to data transformation and the other of which is oriented towards cooperation between different robots.

5.1 Case study 1

5.1.1 Concept

In the first case study, our focus was on the automatic generation of required components from the data definition in the ontology.

5.1.2 Scenario

The goal of this case study was to construct an environment in which a robot continuously generates its position in the seconds format of latitude/longitude, and another robot (or station) intermittently reuses the information in the degree format (DEG) of latitude/longitude. Accordingly, in this scenario, two object types were defined in the ontology, including PositionInSecond and PositionInDEG. A mapping SecondToDEG, which is a relationship used for the conversion between the above objects, was also involved in the ontology.

Figure 5 A sub-tree of the ontology used in the case study (see online version for colours)

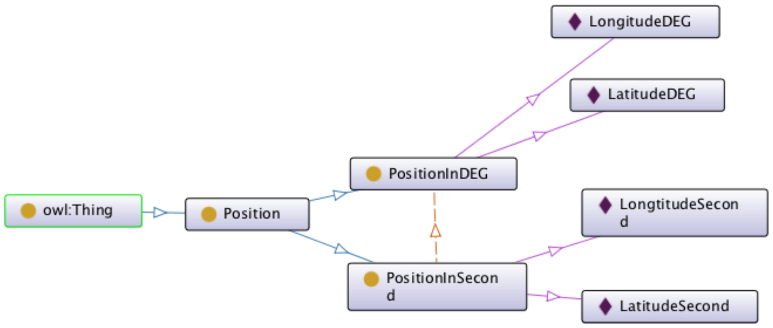
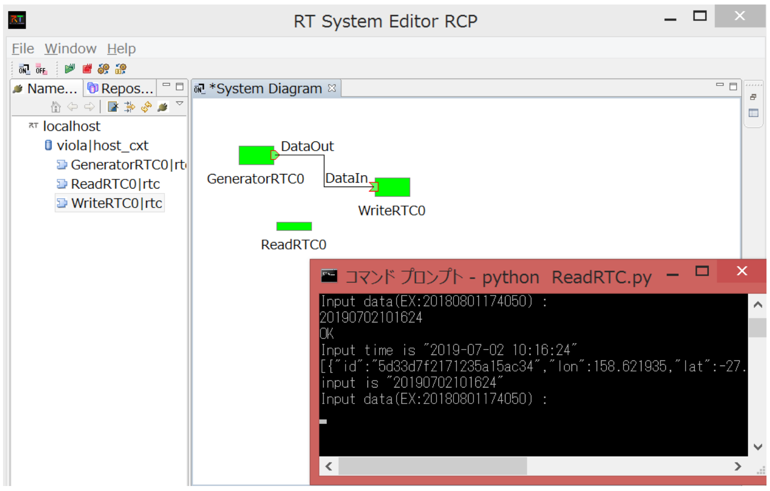


Figure 6 Snapshot of the RTSystemEditor used for the case study (see online version for colours)



```
Input data(EX:20180801174050) :
20190702101624
OK
Input time is "2019-07-02 10:16:24"
[{"id":"5d33d7f2171235a15ac34","lon":158.621935,"lat":-27.
input is "20190702101624"
Input data(EX:20180801174050) :
```

5.1.3 Experiment

The ontology data used for this case study is shown in Figure 5. (in fact, this is a sub-tree of the ontology, which represents data involved in the project as a whole). The framework automatically generated the following parts based on the specification of the objects in OWL (XML):

- WriteRTC: This is an RTC that receives objects of PositionInSecond and sends them to the cloud base station.
- ReadRTC: This is an RTC that obtains objects of PositionInDEG by the use of specified keys from the cloud base station.
- PositionInSecondEntity.java: This is an entity class of PositionInSecond. The corresponding PositionInSecondRepository class accompanies.
- PositionInDEGEntity.java: This is an entity class of PositionInDEG. The corresponding PositionInDEGRepository class accompanies.
- WriteService.java: This is a service class used to receive objects of PositionInSecond from the client system and then save them into the corresponding repository.
- ReadService.java: This is a service class used to obtain objects of PositionInDEG via keys obtained from the corresponding repository and then return them to the client system.
- SecondToDEGService.java: This is a service class used to convert objects of PositionInSecond to the corresponding objects in PositionInDEG.
- Other items: These include other parts, including controllers and an application class, which are used to activate and operate the system on the cloud.

After these parts were created, the framework deployed and activated the application that is accompanied by the components for web and database layers. To perform the experiment, an additional RTC (named GeneratorRTC) that randomly generates positions in the latitude/longitude seconds format was created. The GeneratorRTC is used to imitate a sensor device connected to other components in the robot environment. In addition, we slightly modified a ReadRTC so that we can confirm the result sets obtained from the database by using specified keys as standard output to the console at the local station. In fact, the modified ReadRTC can also be connected to other RTCs that need data and make inquiries to the cloud base station.

These three components were located in RTSystemEditor, which is the dedicated integrator for OpenRTM-aist, and then activated in the local base station in order to observe data management. To provide an overview of this case study, Figure 6 shows a snapshot of an RTSystemEditor in which all components have been activated. We confirmed that the position data generated at the robot system (client-side) were stored in the first-order database in the cloud, transformed in accordance with the mapping, and obtained from the second-order database so they could be properly reused at the client side.

5.2 Case study 2

5.2.1 Concept

In the second case study, our focus was on the following important aspects:

- automatic generation from the definition in the program layer
- promotion of reuse of legacy software components in the library

- cooperation between different robots through cloud data.

The first point shows that our architecture and framework maintains options that can be used to construct an environment through programming in a conventional language that is likely to be familiar to the developers. Although the specification of data through ontology is required in order to provide a standard scenario definition method for non-programmers, the coding is also attractive to developers. This can be important because a writing program is much more convenient when it is necessary to define sophisticated transformation processes between different data types for the multistage databases, as well when integrating the AI engines with available libraries.

The second point shows that our framework can export software components that are compatible with existing legacy components that were used in corresponding projects. In this case study, we reused some legacy components previously developed in our project, which involved constructing an RTC software library based on OpenRTM (Ando et al., 2011). That library is expected to contribute to facilitating reliability and productivity, as well as to enhancing cost reductions and component reusability. So far, a total of 139 RTCs for different types of robots have been developed and registered in RTC-Library-FUKUSHIMA (https://rtc-fukushima.jp/) by around ten companies and organisations, as described in Subsection 3.1. Examples of the registered components added thus far include those related to disaster analysis from an application perspective; crawlers, manipulators, and drones from technological perspectives; cameras, radars, gyros, accelerators, and inertial measurement units (IMUs) from sensing perspectives; and image processing and map construction components from algorithmic perspectives. As such, these entries are contributing to the development of robot systems by third parties, as well as helping to foster robot engineers through their use as educational materials.

To demonstrate the third point, two Lego EV3s, which are small crawler robots, were used in this case study. Although the EV3 robot is often considered a child's toy, it is equipped with a wide variety of functions, sensors, and related components. As a result, the robot contributes significantly to the simulation and educational aspects of our library, particularly since it shares a number of common control interfaces with different practical large-scale robots.

5.2.2 Scenario

In this scenario, an EV3 (EV3-A) attempts to traverse a field that contains some obstacles. While crawling, the robot avoids those obstacles using data collected via its ultrasonic sensor. Intermittent changes to its current velocity, as defined by the vector (vx, vy, va), are transmitted along with an attached timestamp to the first-order database on the cloud. The data in the vector format are then transformed into command data represented as (duration, vx, vy, va) in the second-order database on the cloud. Separately, another EV3 (EV3-B) starts from the common origin point and direction and is automatically controlled to follow the movements of EV3-A based on command data received from the cloud. The important point to note here is that the movements of EV3-B, which does not have any sensors, are controlled in real-time by cloud data with just a slight delay (or by batch processing).

5.2.3 Experiment

The framework exported the following items through the specifications of the three classes in Java.

- WriteRTC: This is the RTC that receives EV3-A velocity data and sends the information to the cloud base station.
- ReadRTC: This is the RTC that obtains commands for EV3-B from the cloud base station.
- VelocityEntity.java: This is the entity class of the velocity. The corresponding VelocityRepository class accompanies.
- CommandEntity.java: This is the entity class of the command. The corresponding CommandRepository class accompanies.
- WriteService.java: This is the service class that is used to receive the velocity data from the client system and save that information into the corresponding repository.
- ReadService.java: This is the service class that is used to obtain the command data from the corresponding repository and return that information to the client system.
- TransformerService.java: This is the service class that is used to convert objects of velocity to the corresponding objects in command.
- Other items: These are other parts, including controllers and an application class, which are used to activate and operate the system on the cloud.

Figure 7 Snapshot of an RTSystemEditor in use for saving data to the cloud (see online version for colours)

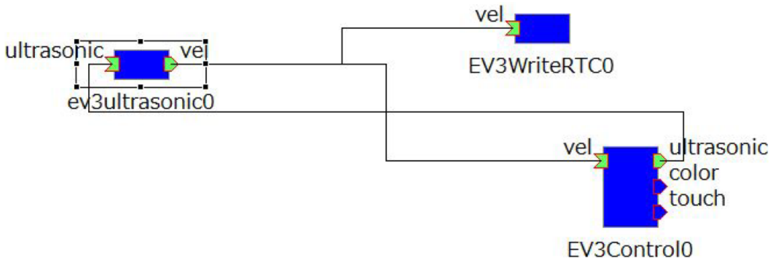
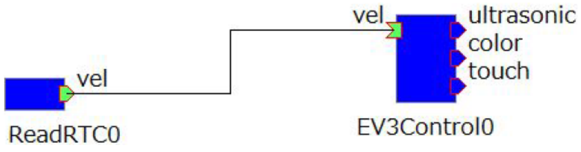


Figure 8 Snapshot of an RTSystemEditor in use for searching data from the cloud (see online version for colours)



The case study reused the following components extracted from the library.

- EV3 control: This is an RTC that has functions for controlling the motors of an EV3 and outputting information from its sensors.

- EV3 ultrasonic: This is an RTC that outputs a velocity vector based on given values of the ultrasonic sensor data obtained from an EV3.

Focusing on EV3-A, in order to demonstrate the case study, Figure 7 shows a snapshot of RTSystemEditor in which components for sending data are activated. In this system, when the RTC detects a velocity change via the ultrasonic sensor value received from EV3-A, it sends the adjusted velocity as (vx, vy, va) along with a timestamp to the WriteRTC (as well as back to the EV3-A). In contrast, focusing on EV3-B, Figure 8 shows the components used for receiving data. Here, we can see that EV3-B is controlled by data from ReadRTC (cloud data).

Note that, in order to provide a more complete explanation, Figures 7 and 8 show the experiment from different perspectives. However, they can be integrated to show that the two EV3s are involved at the same time. Furthermore, although we acknowledge that there is a slight gap between the final positions of the two EV3s, our experiment confirmed that EV3-B could follow EV3-A by obtaining command cloud data that had originally been generated by EV3-A.

6 Discussion

Although the ad hoc applications used in the abovementioned case studies were rather simple, they allowed us to demonstrate that the proposed framework could seamlessly automate the required steps for creating, deploying, and activating components. Therefore, through these case studies, we have also demonstrated that a robot system with data acquisition functions can be constructed automatically via very simple and limited operations and without excessive labour.

In this section, we focus on the productivity, availability, sustainability, scalability, and transparency of our architecture and framework. However, although we will discuss some limitations, we acknowledge that a variety of technical issues related to security, accuracy, and performance levels have not been addressed, because they are out of scope for this paper.

6.1 Productivity

As mentioned above, there are a number of operations that can be laborious if it is necessary for a developer to create a robot system with data acquisition capabilities from scratch. However, by employing our proposed framework, those labors can be drastically reduced. In the abovementioned case studies, the only things the developers needed to do were to select data, define data, or perform transformations. For example, in the second case study, the developer prepared two programs for data and one program for their transformation, after which the framework exported the corresponding two RTCs for the developer on the client side. In addition, on the server side, the framework generated the corresponding two repositories (databases) through entity classes and controllers that bridge the RTCs and the repositories, and then deployed them on the server. Furthermore, the framework launched the cron process to transform data. In summary, based on a limited number of items, the framework automatically generated a number of required components for both the client and server sides.

6.2 Availability

One of the attractive points of our framework is that it exports reusable software components that are compatible with existing legacy components used in corresponding projects. This means that even though the data format, communication interface, and host information will need to be adjusted by changing the system configuration and parameters for each project, the exported components can be directly connected to other components. This is because the interoperability of different components on the distributed system is based on the middleware. On the other hand, since data from any of the stages on the RDR can be obtained through the RESTful API, it is available for other artificial information systems in addition to the target robot environment.

6.3 Sustainability

The proposed framework is based on a technology-independent architecture that is not expected to be significantly affected by future trends. This means that developers need not be biased toward any particular platforms and programming languages. As shown through the first case study, at the highest user interface, the developer can define their data through ontology with a simple GUI. Here, it should be noted that the developers do not need to understand the construct in XML format because they can create and manipulate ontology data graphically in the tree structure. Optionally, they can also define the data and perform mapping through a class and function, respectively, in dedicated modern programming languages. Furthermore, exported components for both the client and server sides can be changed depending on technological trends in the future. In our architecture and related framework, developers can choose the types of components to be used on the client side (RTCs, ROS nodes, etc.) and do not need to consider how the server side is implemented because of the architecture's transparency.

6.4 Scalability

Herein, two case studies were demonstrated through rather simple objects and a toy system in order to focus on validating the mechanisms of the automation process. However, we should also clarify the scalability of our proposed approach. First of all, it should be noted that the ontology defined at the very beginning of a project can include all possible entities and their relationships as a whole. Additionally, for each specific scenario, the system integrator can choose the required object subsets from the ontology in order to develop and deploy the system. Since the scale of the generated components and repositories depends on the selected subsets, a larger system can, in principle, be constructed by using the same framework.

In terms of case studies for demonstrating a practical environment, depending on the availability (interoperability) of the components, the toy systems can be replaced with practical equipment such as large-scale double-armed robots. In fact, large-scale double-armed robots, middle-size crawler robots, and EV3s can all be controlled using a common game controller, which is also registered in our library. Regarding sensors and data, the inputs used in the case studies can be replaced with practical sensor data such as point sets obtained by LiDAR, which can be useful for functions such as creating 3D maps. Another important aspect of the exported components is that they can easily be integrated into simulators alongside other components.

6.5 Transparency

Transparency is one of the most valuable features for examining different perspectives in user-oriented product development. In the environment deployed by the proposed framework, any existing components from the user side can, in principle, transparently access data in the cloud repository through the exported components. Furthermore, since the items generated on the server side are automatically deployed on the cloud, developers will not need to consider the technical details related to implementations, operations, or how the server side is implemented. Thus, developers can focus on the client side system integration by means of standard knowledge representation.

6.6 Limitations

In this paper, we focused on the automation processes used for creating components. However, while we considered the minimum authentication processes necessary for API (controllers), security issues related to the components and their corresponding controllers should be thoroughly considered in terms of network protocols and encryption. Additionally, for discussions related to accuracy and performance, it is important to remember that they depend on the seriousness of the transformers used for the multistage databases. In other words, for the control of autonomous robots (like those used in the second case study), accuracy and performance can be improved by adding or enhancing transformers while considering self-location estimations and SLAM.

7 Conclusions

In this paper, we presented a data acquisition architecture and framework for cloud robotics. In our proposed cloud robotics framework, a robot system can be organised by using a set of reusable legacy software components, as well as special data acquisition components, which then access the RDR on the cloud. These repositories should be organised as multistage databases in order to permit data manipulations between unstructured and structured data. Although there are numerous tasks, the proposed framework can reduce the burden needed to create many of the related software parts, schemas, objects, and settings by automatically generating the required executable codes. As a result, the framework can improve productivity when developing robot systems that actively engage in data acquisition, as well as assist in performing experiments involving valuable information resources. Herein, two case studies were presented to demonstrate that the framework seamlessly generates all of the required items, from data definition to construction, for an operable robot system utilising data acquisition on the cloud. However, while we have demonstrated the productivity, availability, and sustainability of the proposed architecture and framework, there are still issues related to authentication methods and other security issues for deploying services on the cloud that must be resolved. Accordingly, in our future work, we will conduct case studies with mature transformers for real applications while also considering performance and accuracy issues.

References

Al-Masri, E. (2018) 'Enhancing the microservices architecture for the internet of things', *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data)*, pp.5119–5125.

Ando, N., Kurihara, S., Biggs, G., Sakamoto, T. and Nakamoto, H. (2011) 'Software deployment infrastructure for component based RT-systems', *Journal of Robotics and Mechatronics*, Vol. 23, No. 3, pp.350–359.

Beetz, M., Tenorth, M. and Winkler, J. (2015) 'OPEN-EASE – a knowledge processing service for robots and robotics/AI researchers', *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp.1983–1990.

Bergamini, L., Sposato, M., Pellicciari, M., Peruzzini, M., Calderara, S. and Schmidt, J. (2020) 'Deep learning-based method for vision-guided robotic grasping of unknown objects', *Advanced Engineering Informatics*, Vol. 44, p.101052 [online] https://doi.org/10.1016/j.aei.2020.101052.

Bozcuoglu, A.K., Kazhoyan, G., Furuta, Y., Stelter, S., Beetz, M., Okada, K. and Inaba, M. (2018) 'The exchange of knowledge using cloud robotics', *IEEE Robotics and Automation Letters*, Vol. 3, No. 2, pp.1072–1079.

Chen, W., Yaguchi, Y., Naruse, K., Watanobe, Y. and Nakamura, K. (2018a) 'QoS-aware robotic streaming workflow allocation in cloud robotics systems', *IEEE Transactions on Services Computing*, DOI: 10.1109/TSC.2018.2803826.

Chen, W., Yaguchi, Y., Naruse, K., Watanobe, Y., Nakamura, K. and Ogawa, J. (2018b) 'A study of robotic cooperation in cloud robotics: architecture and challenges', *IEEE Access*, Vol. 6, pp.36662–36682, DOI: 10.1109/ACCESS.2018.2852295.

Ezzeddine, M., Morcel, R., Artail, H., Saghir, M.A.R., Akkary, H. and Hajj, H. (2018) 'RESTful hardware microservices using reconfigurable networked accelerators in cloud and edge datacenters', *Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp.1–4.

Filip, I.D., Postoaca, A.V., Stochitoiu, R.D., Neatu, D.F., Negru, C. and Pop, F. (2019) 'Data capsule: representation of heterogeneous data in cloud-edge computing', *IEEE Access*, Vol. 7, pp.49558–49567, DOI: 10.1109/ACCESS.2019.2910584.

Fiorini, S.R., Bermejo-Alonso, J., Gonçalves, P., de Freitas, E.P., Olivares Alarcos, A., Olszewska, J.I., Prestes, E., Schlenoff, C., Ragavan, S.V., Redfield, S., Spencer, B. and Li, H. (2017) 'A suite of ontologies for robotics and automation', *IEEE Robotics Automation Magazine*, Vol. 24, No. 1, pp.8–11.

Funayama, Y., Nakamura, K., Tohashi, K., Matsumoto, T., Sato, A., Kobayashi, S. and Watanobe, Y. (2020) 'Automatic analog meter reading for plant inspection using a deep neural network', *Artificial Life and Robotics*, Vol. 26, No. 2, pp.1–11.

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L. and Bennamoun, M. (2020) 'Deep learning for 3D point clouds: a survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp.1–27, DOI: 10.1109/TPAMI.2020.3005434.

Haidu, A., Bebler, D., Bozcuoglu, A.K. and Beetz, M. (2018) 'KnowRobSIM – game engine-enabled knowledge processing towards cognition-enabled robot control', *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp.4491–4498.

Hildebrandt, C., Kocher, A., Kustner, C., Lopez-Enriquez, C., Muller, A.W., Caesar, B., Gundlach, C.S. and Fay, A. (2020) 'Ontology building for cyber-physical systems: application in the manufacturing domain', *IEEE Transactions on Automation Science and Engineering*, Vol. 17, No. 3, pp.1266–1282.

Hunziker, D., Mohanarajah, G., Waibel, M., D'Andrea, R. and Rapyuta, R. (2013) 'The RoboEarth cloud engine', *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp.438–444.

IEEE Standard Ontologies for Robotics and Automation (2015) *IEEE Std 1872-2015*, pp.1–60.

Inversion of Control Containers and the Dependency Injection Pattern [online] https://www.martinfowler.com/articles/injection.html (accessed 14 October 2020).

Kato, Y., Izui, T., Tsuchiya, Y., Narita, M., Ueki, M., Murakawa, Y. and Okabayashi, K. (2011) 'RSi-cloud for integrating robot services with internet services', *Proceedings of the 37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011)*, pp.2158–2163.

Nargesian, F., Zhu, E., Miller, R.J., Pu, K. and Arocena, P.C. (2019) 'Data lake management: challenges and opportunities', *Proceedings of the VLDB Endowment*, August, pp.1986–1989.

Narita, M., Murakawa, Y., Akiguchi, C., Kato, Y. and Yamaguchi, T. (2009) 'Push communication for network robot services and RSi/RTM interoperability', *Proceedings of the 2009 IEEE International Conference on Fuzzy Systems*, pp.1480–1485.

Niemueller, T., Lakemeyer, G. and Srinivasa, S.S. (2012) 'A generic robot database and its application in fault analysis and performance evaluation', *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.364–369.

Ogitsu, T. and Mizoguchi, H. (2015) 'Practical research of RT-middleware in intelligent vehicles', *International Conference on Connected Vehicles and Expo (ICCVE)*, pp.342–343.

OpenRTM [online] https://github.com/OpenRTM (accessed 14 October 2020).

OpenRTM-aist [online] https://www.openrtm.org/openrtm/ (accessed 14 October 2020).

OWL Web Ontology Language [online] https://www.w3.org/TR/owl-features/ (accessed 14 October 2020).

Park, Y., Choi, J. and Choi, J. (2016) 'A system architecture to control robot through the acquisition of sensory data in IoT environments', *Proceedings of the 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp.749–752.

Popov, V.L., Ahmed, S.A., Topalov, A.V. and Shakev, N.G. (2018) 'Development of mobile robot target recognition and following behaviour using deep convolutional neural network and 2D range data', *IFAC-PapersOnLine*, Vol. 51, No. 30, pp.210–215.

Protege [online] https://protege.stanford.edu/ (accessed 14 October 2020).

ROBOTS [online] https://robots.ieee.org/learn/types-of-robots/ (accessed 14 October 2020).

RTC-Library-FUKUSHIMA [online] https://rtc-fukushima.jp/ (accessed 14 October 2020).

Saha, O. and Dasgupta, P. (2018) 'A comprehensive survey of recent trends in cloud robotics architectures and applications', *Robotics*, Vol. 7, No. 3, p.47.

Spring [online] https://spring.io/ (accessed 14 October 2020).

Tsuichihara, S., Yamaguchi, A., Takamatsu, J. and Ogasawara, T. (2015) 'Using a weighted pseudo-inverse matrix to generate upper body motion for a humanoid robot doing household tasks', *Proceedings of the 2015 IEEE Conference on Robotics and Biomimetics*, pp.333–338.

Waibel, M., Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Lopez, D.G., Haussermann, K., Janssen, R., Montiel, J.M.M., Perzylo, A., Schiessle, B., Tenorth, M., Zweigle, O. and Van de Molengraft, M.J.G.R. (2011) 'RoboEarth – a World Wide Web for robots', in *IEEE Robotics & Automation Magazine*, Vol. 18, No. 2, pp.69–82.

Wang, R., Mou, X., Sun, J., Liu, P., Guo, X., Wo, T. and Liu, X. (2020) 'Cloud-edge collaborative industrial robotic intelligent service platform', *2020 IEEE International Conference on Joint Cloud Computing*, pp.71–77.

Watanobe, Y., Yaguchi, Y., Miyaji, T., Yamada, R. and Naruse, K. (2019) 'Data acquisition framework for cloud robotics', *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, pp.1–7.

Williams, K., Li, L., Khabsa, M., Wu, J., Shih, P.C. and Giles, C. (2014) 'A web service for scholarly big data information extraction', *Proceedings of the 2014 IEEE International Conference on Web Services*, pp.105–112.

World Robot Summit (WRS) [online] https://worldrobotsummit.org/en/ (accessed 14 October 2020).

Yaguchi, Y., Nitta, Y., Ishizaka, S, Tannai, T., Mamiya, T., Naruse, K. and Nakano, S. (2017) ‘Formation control for different maker drones from a game pad’, *Proceedings of the 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp.1373–1378.

Yoshino, D., Watanobe, Y., Yaguchi, Y., Nakamura, K. and Anma, K. (2017) ‘Application possibility of OpenRTM-aist-based integrated robot systems using CORBA interfaces and brokered Pub/Sub messaging interfaces’, *The Proceedings of JSME annual Conference on Robotics and Mechatronics (Robomec)*, pp.2A2–J08.

Yoshino, D., Watanobe, Y., Yaguchi, Y., Nakamura, K., Ogawa, J. and Anma, K. (2018) ‘Provision of remote management infrastructure for RT systems using mosquitto MQTT message broker’, *The Proceedings of JSME annual Conference on Robotics and Mechatronics (Robomec)*, pp.2A1–G08.

Yoshino, D., Watanobe, Y., Yaguchi, Y., Nakamura, K., Ogawa, J. and Naruse, K. (2019) ‘AMQP communication interface on RT middleware for highly-reliable IoR system construction’, *The Proceedings of JSME annual Conference on Robotics and Mechatronics (Robomec)*, pp.2A1–M09.