A Case Study: Robot Manager for Multi-Robot Systems with Heterogeneous Component-based Frameworks

Min Ho Lee[1], Ho Seok Ahn[2], Bruce A. MacDonald[3]
Department of Electrical and Computer Engineering, CARES, University of Auckland
Auckland, New Zealand
[1]mlee242@aucklanduni.ac.nz, [2, 3]{hs.ahn, b.macdonald}@auckland.ac.nz,

Abstract

Robotic software developers may need to use different component-based software frameworks, such as ROS, OPROS and OpenRTM, for developing intelligent robots, but then face challenges with interoperability, code maintainability and reusability over multiple frameworks. In this paper, we present a design and implementation of the robot manager software for our robotic software framework, which helps to integrate different programming frameworks easily, and to minimize the impact of different frameworks and new versions. We have designed the robot manager to fulfil the following four specifications: interoperability, compatibility, heterogeneous systems, and Application Program Interface (API) abstraction. The proposed framework allows us to use different applications with various components from different frameworks. It also supports communication between heterogeneous robot systems. For an evaluation case study, we have developed two different robot systems, which cooperate with each other using our robotic software framework, which in turn integrates components from different frameworks. We report on our evaluation of the effectiveness of the four design specifications of the robot manager.

1 Introduction

Building intelligent service robotic system requires integration of many technologies such as image processing, speech processing, precise control, and sensor fusion. Developing each technology takes considerable development effort, but integration may require significantly more effort [Ahn et al., 2008]. To reduce these efforts, we use component-based software frameworks [Bruyninckx et al., 2013] such as ROS [Quigley et al., 2009], OPROS [Jang et al., 2010], Open-RTM [Noriaki et al., 2008], and ROCOS [Jayawardena et al., 2012]. These provide a modular approach to building complex robotic systems in terms of functionalities, libraries and tools created as software modules, allowing developers to easily reuse theirs and others' code.

Fig. 1. Multi-robot systems with the UoA robotic software framework. The left robot is a receptionist robot to direct patients to the required healthcare services. The right robot is an assistant robot help patients to measure vital signs data, such as blood pressure, blood oxygen saturation and pulse rates.

However, researchers must re-write intelligent components and applications in different ways to transfer them from one framework to another [Lee et al., 2014a]. Some researchers have developed techniques for interoperating between two frameworks, such as ROS-OpenRTM [Biggs et al., 2010] and ROS-OPROS [Jang., et al., 2012]. But these methods provide only one-to-one interaction between the frameworks by translating the messages through custom bridging software. Furthermore, in order to develop these bridges, the developer is still required to understand the concepts of both frameworks.

Recently, there has been increasing interest in formalizing the process for robot development. YARP is a library that provides communication protocols for different robotic applications [Metta et al., 2006]. However beyond providing communication protocols to be used between the frameworks, YARP is limited in that the semantic of mapping different frameworks must be defined manually. Rosbridge [Crick et al., 2012] uses a generic WebSocket/JSON interface to communicate with the ROS environment. However, the developer must understand the semantics of the ROS framework. In our previous research, we have designed a University of Auckland (UoA) Robotic Software Framework, which helps integrate different programming frameworks easily, and minimizes the impact of framework differences and new versions. We have designed the high-level API for the application, which does not require modification of the application when it calls different components from different software frameworks [Lee et al., 2014a]. Also, we can use the same API for multiple applications written in different programming languages. The UoA Robotic Software Framework supports connections to external systems, such as smart home systems and our sensor manager system that manages various sensor devices [Lee et al., 2014b].

In this paper, we focus on the design and use of the robot manager system, which is a core component of the UoA Robotic Software Framework. The robot manager orchestrates the connections between different applications and different frameworks. The robot manager translates the requests from applications using the designed API and calls appropriate components of different frameworks based on a configuration file, which includes a user-defined priority table of components. It makes it easy for a developer to build intelligent robot systems while minimizing code changes. The robot manager also supports communication with other systems, such as robots and environmental systems. We have developed a case study with two different robots that cooperate to help people, and we have evaluated the effectiveness of the proposed system.

This paper is organized as follows. In Section 2, we introduce our robotic software framework. In Section 3, we explain the robot manager's four modules. We present the system integration in Section 4, and evaluations in Section 5. Finally, we conclude this paper in Section 6.

2 UoA Robotic Software Framework

2.1 Motivation

The motivation of our framework is to increase reusability of components of different frameworks allowing the different researchers and developers to easily utilise works done by other people. There a number of frameworks used by multiple groups, such as OROCOS, OPROS, OpenRTM, and ROS, each following its own protocols and standards, yet with many similarities. While there are sets of standards for robotic framework protocols such as the RT group, which OpenRTM is based on, other frameworks such as ROS are not standardized [Noriaki et al., 2005]. In addition, each framework runs in a specific operating environment, for example, ROS runs mainly on Ubuntu/Linux, and ROCOS runs mainly on Windows. With different organizations using their own frameworks, components developed in one framework must be rewritten for the component to be used in another framework. This can be quite difficult and requires considerable development effort, as mentioned above.

A problem with using frameworks is that they often turn into legacy systems; the software continuing to be used, despite the availability of newer technologies and more efficient methods of performing a task. It is often difficult to support backward compatibility from new versions of the frameworks. Therefore with legacy code, the developers will be either restricted to using the old framework or required to resource the effort of rewriting all components to suit the new framework.

Each framework consists of different semantics and therefore, has a different associated learning curve. In order to rewrite the components to work in a different framework, developers will be required to learn the different semantics, concepts and functionalities associated with the new framework, and also may be required to map these concepts together in order to provide the same component services as in the original framework. For example, OpenRTM has a built-in Finite-State Machine (FSM) mechanism. If a component from OpenRTM must be translated into ROS, then the FSM feature must be implemented in the process of translation. Such semantic mapping can take a lot of development effort and therefore, can be problematic

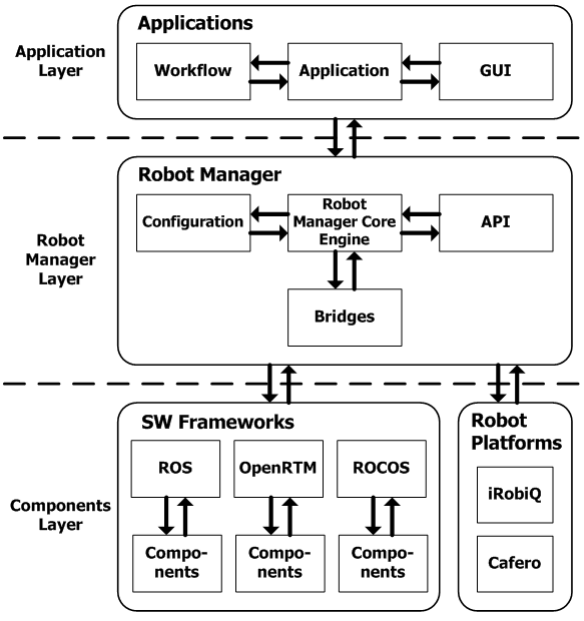

Fig. 2. Architecture of the UoA Robotic Software Framework. It consists of three layers; an application layer, a robot manager layer, and a components layer that includes various frameworks.

when a large number of features from one framework must be used in integration with another framework.

2.2 Concept of the Framework

The main goal of our framework is to allow interoperation between existing component-based frameworks and reduce the effort of having to use multiple frameworks. For our design, we must consider various aspects such as usability and scalability that not only allow developers to control robot hardware using components from any frameworks, but also allow developers to easily add components of other frameworks in the future. In addition, it is also important to define methods to dynamically classify and monitor the connected robot frameworks and the data channels used. For this, our design includes the following functions.

2.2.1 Interoperability

The robotic application should be able to choose components from any available framework connected to the robot manager and use its component services. Also, the robot manager is required to determine the compatibility for any services that have dependent components.

2.2.2 Compatibility

The robot manager framework should not be restricted to a specific platform or environment but rather should function in multiple platforms and environments. By allowing interoperation of multiple frameworks, we can effectively eliminate the need for creating several versions of components to match the requirements of different frameworks.

2.2.3 Heterogeneous system

Any kind of application should be able to access our framework from any operating environment. This also requires the communication protocol to be implemented for any languages and any operating environment. This includes supporting multi-robot systems.

2.2.4 API abstraction

The APIs should provide a method for applications to use components without having to rely on component specific feature names, so that if the application needs to use the same function in a different framework, the application software wouldn't require code modification. By providing a high level API, we are able to reduce the user's effort of having to learn each framework's semantics.

2.3 Design of Framework

Fig. 2 shows the UoA Robotic Software Framework architecture, which consists of three layers; an application layer, a robot manager layer, and a components layer [Lee et al., 2014a]. In the application layer, various applications are executed based on workflows. The application is able to interact with the robot manager using a predefined API and a combination of WebSocket connections and JSON messages [Lee et al., 2014a] [Crick et al., 2012]. The robot manager is responsible for monitoring, mapping and channelling the communication among the components of different frameworks. The robot manager uses bridges for communicating with components. The component layer includes software frameworks, robot platforms, and external systems such as home automation systems and a sensor system. The components of different frameworks are able to provide interaction with the robot manager layer through bridges, which provide translations of the internal communication of the framework to a generic format and vice versa.

3 Robot Manager

The robot manager provides three key functionalities; 1) managing status of each connected framework, 2) mapping a higher level API to framework components, 3) transferring data among frameworks. In order to cover these functionalities the robot manager is divided into four modules each implementing solutions to respective functionalities: a robot manager core, API, a configuration file, and bridges.

3.1 Robot Manager Core

The robot manager core is responsible for managing the status of each connection, and setting up the communication channel for the application to use. This is done by a map of connections; a connection will be registered once the communication channel is established and ready to be used. It also keeps track of the run-time status of each bridge and which dataflow belongs to which bridge. The messaging in the robot manager layer uses JSON format as a generic form of communication between the frameworks.

3.2 API Design

In order to provide the applications with a high level abstract API, we have designed API sets and protocols for the UoA Robotic Software Framework. We have based our design on an analysis of API structures and protocols of existing systems. For our version of the API, we have chosen three different robotic frameworks to analyse; ROS, ROCOS, and OpenRTM. The key required elements were arguments that establish a data port, and the message types required for input and output. Another reason we chose these elements was because each framework has its own unique representative model and standards; ROS using a home-made channel model, OpenRTM following the RT standard using CORBA/ICE for communication and ROCOS using plain messaging.

3.3 Configuration File

The configuration file is responsible for mapping the higher level API to the framework-specific API. The configuration file contains a logical table implemented in XML and listing available robotic components for each robotic framework. Each table entry specifies priorities, required input arguments and outputs to each component

that is connected to the bridge, in order to enable the connection to the components to be established. Finally, the bridge is responsible for translating from the message format used in each framework into generic JSON format. The bridge translates the messages by consuming the generic JSON messaging to establish a connection and send data to the specific framework. In addition, it is able to do the reverse operation turning the framework specific data into the generic JSON format to be later handled by the robot manager core.

3.4 Bridges

The bridges are responsible for two main functions; finding out if a requested component exists within the framework, and translating the generic robot manager protocols to the targeted framework protocols. The process will be further explained in the next section regarding interoperation of the components. The bridges implement the semantics of the respective frameworks such that, from the application's perspective, it is shown as a simple input and output socket port that can be used to receive or send data flows. Therefore, through interoperation, the application/component can use another framework's component without having to know about the semantics of that framework. The bridges are also created to be modular and can be placed running on a different machine from the central robot manager core. Furthermore, while the bridges have been implemented in Java for our experimental purpose, they can be implemented in any other programming languages on any platforms as long as they have the ability to support the communication logic used by the robot manager.

3.5 Interoperation of components

Fig. 3 shows the overall procedure of interoperation between an application and frameworks. If the robot manager receives a request message for initialization from the application, it checks dependencies using the API-frameworks map in the configuration file. Then, the robot manager goes through the validation cycle checking the compatibility of input of the depending element and the output of the dependent component. From this process, we can update the API-frameworks map with valid connections.

When the application calls the API to execute some intelligent module, the robot manager establishes appropriate connections based on the map in the configuration file. For example, when we call the API for a face detection module, the robot manager finds the camera image output and face detection component, which are able to communicate with each other, by checking message structures between the input and the output of component dependencies. The required dependencies are defined in the configuration file. If robot manager finds an appropriate combination, it connects them. Otherwise, it returns a failure response to the application. The robot manager can establish and manage multiple connections at the same time.

When the connection is established, a component can use the data output until it is disconnected. The robot manager checks the connections regularly, and reports the status to the application. For example, if the speech

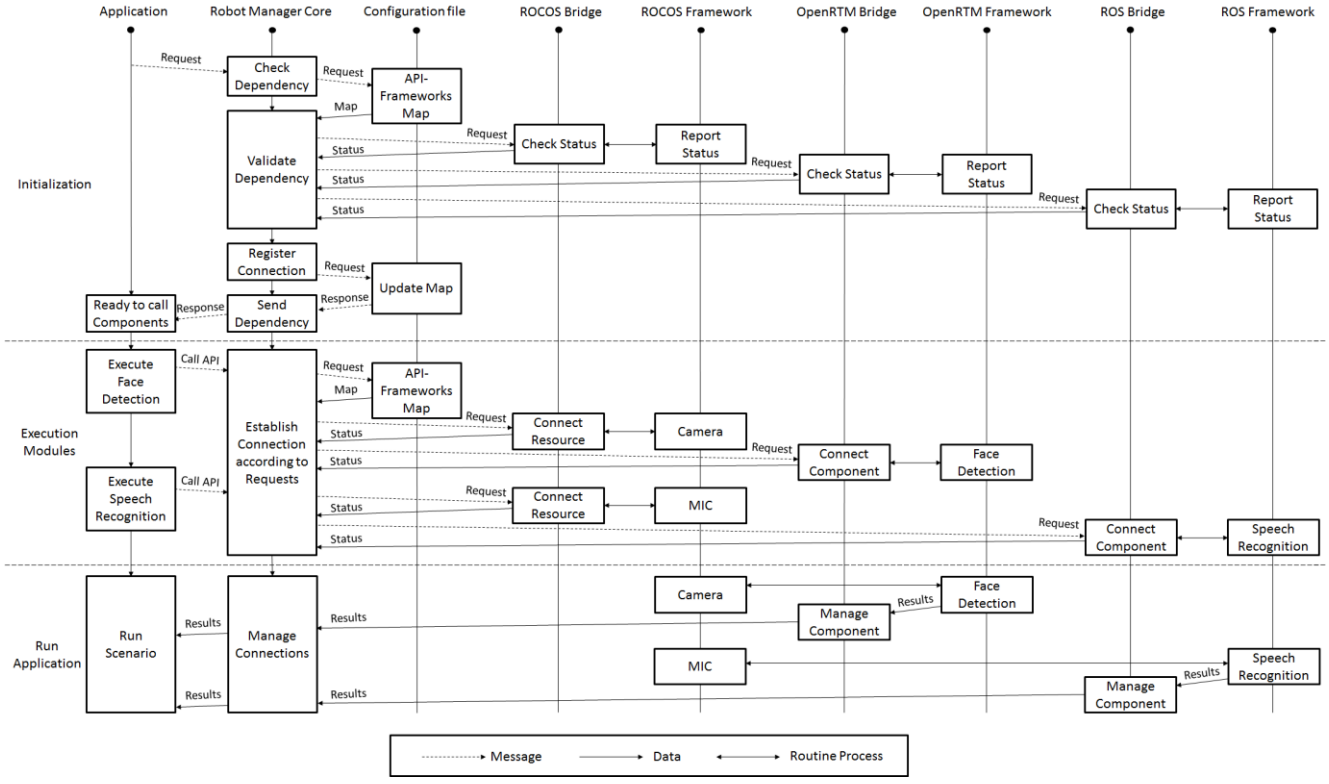

Fig. 3. Overall procedure of interoperation between an application and frameworks. The request from the application goes to the appropriate framework through the robot manager and related bridge.

recognition component of the ROS framework recognizes what the human user said, the ROS bridge sends the result to the application through the robot manager.

4 System Integration

We have developed a multi-robot system using two robot platforms: iRobi and Cafero, which are shown in Fig. 1. Both robots are developed by Yujin Robot in South Korea. The iRobi robot is smaller in size measuring 45x32x32cm and weighing 7 kg. iRobiQ has a child voice using Yujin's voice generation engine. The role of the iRobi robot has been designed as a receptionist robot to direct patients to the required healthcare services.

The Cafero robot is a kiosk type robot platform 1.2 metres high, also developed by Yujin Robot [Ahn et al., 2015]. It has a tiltable touchscreen, microphones, ultrasonic sensors, bumper sensors, and a laser range finder. A touch screen helps interactions by showing messages or pictures and accepts inputs through the touch screen. The role of the Cafero robot has been designed as an assistant robot to help patients to measure vital signs data, such as blood pressure, blood oxygen saturation and pulse rate.

Fig. 4 shows the system diagram of two robot systems using different kinds of components from different services. We have developed the iRobi application using Flex/ActionScript 3.0. iRobi uses four components from three SW frameworks; speech recognition from ROS, face detection from OpenRTM, and speech generation and sound direction detection from ROCOS. We developed the assistant robot application using HTML5 and Javascript. The Assistant robot uses two components from two SW frameworks; face detection from OpenRTM, and speech generation with Festival voice generation engine from ROCOS.

5 Evaluations

To evaluate how well our framework satisfies the four key specifications, we have measured the performance of our framework. This also allows us to evaluate how applicable our framework is in robotics applications that often require real time performance in terms of communication and processing. To measure the performance, the latency of each layer was measured by taking timestamps at each interface. The timestamp functions used were native to the respective language used in the development, such as Java, JavaScript, and ActionScript, and was measured to the nearest millisecond that JavaScript provided. For the receptionist robot, both OpenRTM and ROCOS ran inside the iRobi robot while ROS was running on a separate Linux laptop and was wirelessly communicating with the robot to provide speech recognition. Likewise for the assistant robot, both of the used frameworks ran on the Cafero robot to provide its vital sign measuring service and for

TABLE I. Shows various software programs used in the development. While all these components have their own specific software requirements, we were able to integrate them using our UOA Framework to create our robot systems (assistant robot and receptionist robot) without having to change the existing code to allow interoperations.

Comoponents	Framework	Targetted OS Platform	Implementation Language
Face Detection	OpenRTM	Windows	C++
Speech Recognition	ROS	Linux	Python (the wrapper) C++ (Engine)
Speech Generation	ROCOS	Windows	C++(Wrapper)
Healthcare Measuring Devices	ROCOS	Windows	C++
Receptionist Robot Application	---	Windows	Flex/ ActionScript
Assistant Robot Application	---	Platform Independant	HTML/ JavaScript

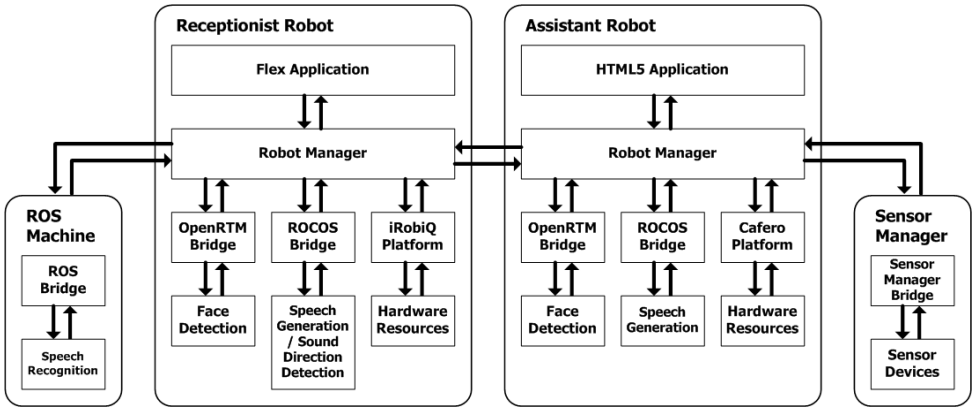

Fig. 4. System diagram of a multi-robot system. A receptionist robot uses four components from three software frameworks, ROS, OpenRTM, and ROCOS based on the iRobi robot platform. An assistant robot uses two components from two software frameworks, OpenRTM and ROCOS based on the Cafero robot platform.

```
<API name="SpeechText">
<!--for ROCOS-->
  <component name="PlayTTS" dependency="" Package="ROCOS">
    <inputs>
      <param name="text" datatype="service" type="PlayTTS"/>
    </inputs>
    <outputs>
    </outputs>
  </component>
  <component name="Festival" dependency="" Package="ROCOS">
    <inputs>
      <param name="text" datatype="service" type="FestivalSay"/>
    </inputs>
    <outputs>
    </outputs>
  </component>
</API><!--Speech recognition-->
<API name="speechToString">
<!--for ROS-->
  <component name="/recognizer/output:std_msgs/String"
dependency="" Package="ROS">
    <inputs>
    </inputs>
    <outputs>
      <param name="Text" type="std_msgs.String"
argument="/recognizer/output" dataType="dataout"/>
    </outputs>
  </component>
</API>

<!--FaceDetection-->
<API name="FaceDetection">
<!--for OpenRTM-->
  <component name="Point2D" dependency="" Package="openRTM">
    <inputs>
    </inputs>
    <outputs>
      <param name="point" type="Point2D" dataType="dataout"/>
    </outputs>
  </component>
</API>
```

Fig 5. Shows portion of configuration XML that was used to define the functionalities used in the applications. The applications developed in the robots uses the same high level API which abstracts the lower level framework-specific function calls.

detecting the face of the patients.

As a result, the robots acted according to the designed workflow providing services as requested. In a local environment, the latency of each layer and the overall layer was less than 1 millisecond and was constant throughout the delivery of the component service, showing the potential to carry out real-time services. For wireless communication, the latency has shown a wide range from 10ms to as large as 1 second. However, this latency is likely due to the overhead of using wireless communication. This just means that our framework requires some kind of real time control mechanism to allow messaging to be done in real time. Otherwise our experiments have shown how our key specifications have been met.

5.1 Interoperability

Through our scenario study, we have integrated components from different existing robotics frameworks: ROS, OpenRTM and ROCOS. As required by the work flow, the robots were able to detect human faces using OpenRTM's face detection component, and to speak predefined sentences using ROCOS's speech generation component. As an additional feature, receptionist robots were able to communicate with human speech through ROS's speech recognition and give feedback according to the workflow. The assistant robot was able to use a vital signs measuring component from ROCOS. The robot manager was able to both subscribe (receive the data) and also publish (output data) from each of the available frameworks using a common semantic interface. The data from each of the components' outputs was processed by our robot's application. From the results, we confirmed that the robot manager framework is able to fulfil the criteria of interoperability by using multiple components from different frameworks. However we have yet to evaluate the dependency checking of the frameworks.

5.2 Compatibility

We developed two different applications, which operate in two different environments; the Flex application (executed solely in the Windows environment), and the HTML5/ JavaScript application (executed in web browsers in both Windows and Linux environments). Both applications are used to implement the control logic of the robotic systems through our framework by connecting to the robot manager and accessing components. This interaction was allowed due to our use of a generic interface (WebSockets and JSON messaging) which is included in many software development languages and environment, allowing simple integration between frameworks. The robot manager is also able to provide services that are derived in different environments such as a speech recognition service from the Linux environment. Therefore, we have confirmed that our framework is able to fulfil the requirement of the second design concept, compatibility. In addition, we are able to achieve reusability of the configuration file as our both of our software applications utilise the same configurations for any common components. In our experiment, both the Cafero robot and the iRobi robot utilise the face detection component from OpenRTM and therefore use the same configuration data in the configuration file.

5.3 Support for heterogeneous platforms

Our implementation of robot manager allowed us to utilise existing components and applications created on various platforms. Components are usually restricted by their software platform due to the dependency that is usually associated with the platform. However by using our framework we were able to create our robotic system without having to change the existing code of each component. Table I shows the components and their software specification. While some components have different specifications we were able to create different applications without having to add any extra code modification.

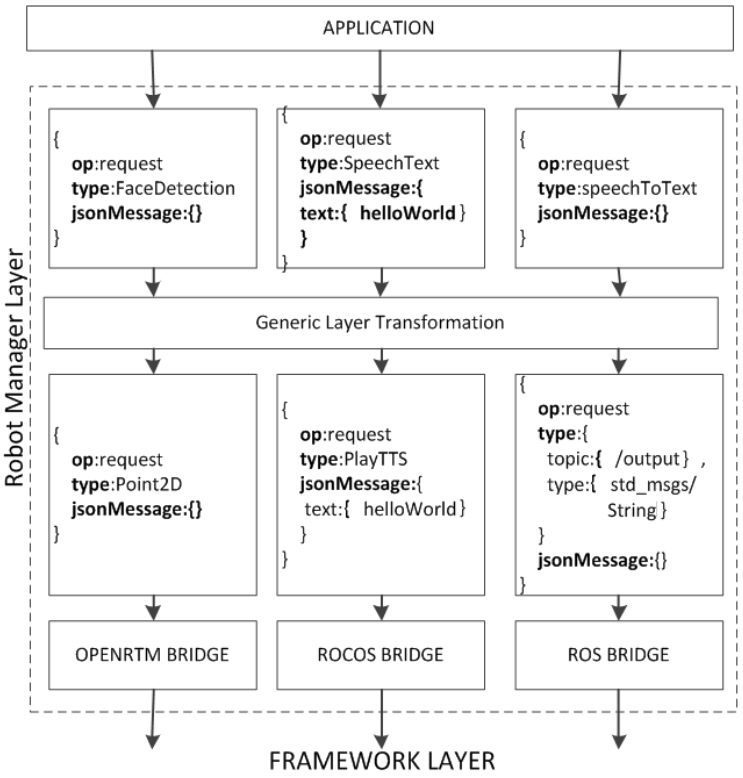

Fig 6. shows message transformation from a generic message to a framework-specific protocol. On receiving generic messages from the application layer, the robot manager transforms the generic message into an intermediary message containing relevant information that can connect to each framework using the configuration file. These messages are then passed to the bridge component of the robot manager layer which then transforms the messages to framework-specific messages.

5.4 API Abstraction

We have provided API abstraction through using our configuration table that maps the high level API name. Fig 5 shows a portion of the XML configuration file which allows the generic communication to occur using JSON messaging. These messages do not contain additional parameters required by each framework, allowing each of the function calls to be independent of the framework of the components. These additional parameters are all defined in the configuration file, which focuses on the idea of separation of concerns, a key idea in reusability. Fig 6. shows the experimental scenario procedure and shows how the message is transformed from generic protocols, which is independent of the additional parameters used to connect with the corresponding frameworks. Through this API abstraction we are able to achieve reuse of application software by using different types of speech functions. The Cafero robot uses a New Zealand accented voice as text-to-speech implemented in Festival while iRobi uses the speech generation engine developed by robot manufacturer, Yujin. Through the use of our framework, we are easily able to change voices without changing the code, as the same interface is used, which reduces the effort for application developers. From these results, we are able to confirm that our UoA Robotic Software Framework satisfies the last design concept, API abstraction.

6 Conclusions

In this paper, we have described the design and implementation of the UoA Robotic Software Framework that allows applications to use multiple components from different frameworks and use them in interoperation. Our framework design incorporates the design of APIs, control procedures and architecture in order to fulfil the four key specifications including interoperability, compatibility, heterogeneous system, and API abstraction. We have applied our framework to a multi-robot application with two kinds of healthcare robot systems, which have been implemented in different programming languages and use components from different frameworks. From the experiment, we have confirmed that the proposed framework satisfied the four design concepts, which enables researchers and developers to save effort when building intelligent robots. The current limitation of our framework is that it only supports WebSockets as the transport mechanism which can be quite slow if each message were to be large in size. To address this issue, we are planning to add more types of transport mechanisms such as TCP or UDP which will help the applications in meeting their operational requirements. We are also planning to apply other software evaluation methods such as software reusability metrics to analyse the quantitative benefit of our system and ensure that the benefits are delivered. This could also be compared with other existing frameworks. Finally, we plan on extending our framework by developing more bridges to allow applications to use more component services. Also, we will develop an easy to use programming toolkit for our framework to enhance usability. Future work such as usability trials will show the benefits to developer's effort in creating robotic applications. A usability test can be designed by first analysing the usability experience of existing frameworks, which could be compared with the usability experience of our framework. This will allow us to identify and compare different learning curves associated with using different frameworks and whether the use of our framework simplifies the process of accessing component functionalities. The project can be made into an Open Source project, once such validations are made.

References

[Ahn et al., 2008] Ho Seok Ahn, Young Min Baek, In-Kyu Sa, Jin Hee Na, Woo-Sung Kang, and Jin Young Choi, "Design of Reconfigurable Heterogeneous Modular Architecture for Service Robot," In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pp. 1313-1318, 2008.

[Ahn et al., 2015] Ho Seok Ahn, Chandan Datta, I-Han Kuo, Rebecca Stafford, Ngaire Kerse, Kathy Peri, Elizabeth Broadbent, Bruce A. MacDonald,

"Entertainment Services of a Healthcare Robot System for Older People in Private and Public Spaces," In Proceedings of the 2015 International Conference on Automation, Robotics and Applications, 2015.

[Biggs et al., 2010] G. Biggs, N. Ando, T. Kotoku, "Native Robot Software Framework Inter-operation," Simulation, Modeling, and Programming for Autonomous Robots, pp. 180-191, 2010.

[Bruyninckx et al., 2013] B. Herman, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmarthemis, L. Gherardi, and D. Brugali, "The BRICS component model: a model-based development paradigm for complex robotics software systems," In Proceedings of the Annual ACM Symposium on Applied Computing, pp. 1758-1764, 2013.

[Crick et al., 2012] Crick, Christopher, Graylin Jay, Sarah Osentoski, and Odest Chadwicke Jenkins, "ROS and Rosbridge: roboticists out of the loop," In Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction, pp. 493-494, 2012.

[Datta et al., 2013] Chandan Datta, Hong Yul Yang, I-Han Kuo, Elizabeth Broadbent, and Bruce A MacDonald, "Software platform design for personal service robots in healthcare," In Proceedings of the IEEE International Conference on RAM, pp. 156-161, 2013.

[Jang et al., 2010]Choulsoo Jang, Seung-Ik Lee, Seung-Woog Jung, Byoungyoul Song, Rockwon Kim, Sunghoon Kim, and Cheol-Hoon Lee, "OPRoS: A New Component-Based Robot Software Platform," ETRI Journal, Vol. 32, no. 5, pp. 646-656, 2010.

[Jang., et al., 2012] C. Jang, B. Song, S. Jung and S. Kim, "A heterogeneous coupling scheme of OPRoS component framework with ROS," In Proceedings of the RAS/EMBS International Conference on Ubiquitous Robots and Ambient Intelligence, pp. 298-301, 2012.

[Jayawardena et al., 2012] C. Jayawardena, I. Kuo, C. Datta, R. Q. Stafford, E. Broadbent, and Bruce. A. MacDonald, "Design, implementation and field tests of a socially assistive robot for the elderly: HealthBot Version 2," In Proceedings of the RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics, pp. 1837-1842, 2012.

[Lee et al., 2014a] Min Ho Lee, Ho Seok Ahn, and Bruce MacDonald, "Design of an API for Integrating Robotic Software Frameworks," In Proceedings of the 2014 Australasian Conference on Robotics and Automation (ACRA 2014), 2014.

[Lee et al., 2014b] Min Ho Lee, Ho Seok Ahn, Kevin Wang, and Bruce MacDonald, "Design of a Healthcare Sensor Managing System for Vital Sign Measuring Devices," In Proceedings of the 2014 International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014), pp. 519-530, 2014.

[Metta et al., 2006] Metta, Giorgio, Paul Fitzpatrick, and Lorenzo Natale, "YARP: yet another robot platform," International Journal on Advanced Robotics Systems, vol. 3, no. 1, pp. 43-48, 2006.

[Noriaki et al., 2005] Ando, Noriaki, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku, and Woo-Keun Yoon, "RT-middleware: distributed component middleware for RT (robot technology)," In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pp. 3933-3938, 2005.

[Noriaki et al., 2008] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku, "A Software Platform for Component Based RT-System Development: OpenRTM-Aist," Simulation, Modeling, and Programming for Autonomous Robots, pp. 87-98, 2008.

[Quigley et al., 2009] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger R. Wheeler, and A. Ng., "ROS: an open-source Robot Operating System," In Proceedings of the ICRA workshop on open source software, 2009.