# 論文 / 著書情報
# Article / Book Information

| | |
|---|---|
| Title | Normalizing abstractions of heterogeneous robotic systems by using Roles: usability study in the administration of software and development tools |
| Authors | Ceron Lopez A, Fukushima E, Endo G |
| Citation | Advanced Robotics, , , |
| Pub. date | 2016, 2 |
| Note | This is an electronic version of an article published in ADVANCED ROBOTICS, , , , 2016. ADVANCED ROBOTICS is available online at: www.tandfonline.com/Article DOI; http://dx.doi.org/10.1080/01691864.2016.1142896. |
| Note | このファイルは著者（最終）版です。<br>This file is author (final) version. |

FULL PAPER

# Normalizing Abstractions of Heterogeneous Robotic Systems by using Roles: Usability study in the Administration of Software and Development Tools

Arturo E. Cerón López[a*], Edwardo F. Fukushima[b,a] and Gen Endo[a]

[a] *Department of Mechanical and Aerospace Engineering, Tokyo Institute of Technology, Tokyo, 152-8552 Japan*; [b] *Department of Mechanical Engineering, Tokyo University of Technology, Tokyo, 192-0914 Japan*

(*Received AA BB CC; accepted XX YY ZZ*)

The modern approach for developing software in service robotics is through the use of robotics middleware platforms. Although the main concept is similar across platforms, their discrepancy among abstractions hinders the composition and administration of highly integrated systems, resulting in usability issues for the developers. We identify that this can be improved by normalizing the abstractions of the heterogeneous systems at runtime. We propose a framework that sets the concept of "Roles" to define a novel method of creating and reasoning system models by normalizing abstractions to produce practical platform-agnostic representations of systems, which is implemented in a cross-platform infrastructure and a GUI. This paper verifies and validates the functionality and usability (by benchmarking) of the proposed framework through a case-study using a real and a simulated mobile robot. The results from usage trials with test subjects showed improved usability, and demonstrated the overall advantage of using the proposed approach.

**Keywords:** robotics middleware; roles; cross-platform; administration; usability

## 1. Introduction

Service robots' application domain is broader than industrial robots; according to ISO-8373, they are autonomous actuated machines that perform useful tasks for humans or equipment, excluding industrial automation. Since their required capabilities are usually developed independently in the respective research fields, their software composition tends to be heterogeneous. Due to this, there have been various efforts at making the heterogeneity feasible (i.e. middleware, cross-platform interfaces, and formal component models). However, the impact this has on usability when the developers perform administration tasks in such heterogeneous and highly integrated robotic systems has not been thoroughly analyzed. Defined in accordance with ISO-9241, usability is: the effectiveness, efficiency and satisfaction with which the users (i.e. developers) can achieve specified goals (i.e. compose and administer robotic software systems) in particular environments (i.e. the development platforms). We identified that this kind of usability can be achieved through the normalization at runtime of the different abstractions employed by each software package and their development tools. This is valuable during the prototyping stage of a robotics application to increase the usability of the involved software and tools, allowing the developers to unhesitatingly work with the systems and focus on the actual applications that they want to build and test.

Moreover, by having a method of normalizing software's abstractions at runtime, the creation of a global knowledge database that curates the best solutions for robotics software becomes possible, indicating which software elements and platforms (and associated dependencies) are

*Corresponding author. Email: aceron@robotics.mes.titech.ac.jp

Table 1. List of Robotics Middleware projects with development dates

| Projects | Year | Projects | Year |
|---|---|---|---|
| NASREM | 1980s | MS. Robotics Dev. Studio, MARIE, RSCA, | 2006 |
| OPEN-R, SmartSoft | 1990s | Sensor Data Proc. Middleware, WURDE | |
| CLARAty, Player/Stage | 2000 | Honda's Dist. Humanoid Robots Middleware, | |
| MIRO, OROCOS, ORiN, | 2002 | KAIST's Middleware, KOMoR, PEIS Kernel, | 2007 |
| ERSP, YARP, Pyro | | OPRoS, ASEBA, ROS | |
| URBI | 2003 | LabView (Robotics Module) | 2009 |
| ORCA, Webots | 2004 | MOOS-IvP | 2010 |
| JAUS, CAMUS, RT-Middleware, | | CoHoN, Kukanchi | 2011 |
| iRobot AWARE, RoboFrame, | 2005 | Remote Data Transm. Middlew. for Telerob., DARC | 2013 |
| UPnP Robot Middleware | | HAMSTER | 2014 |

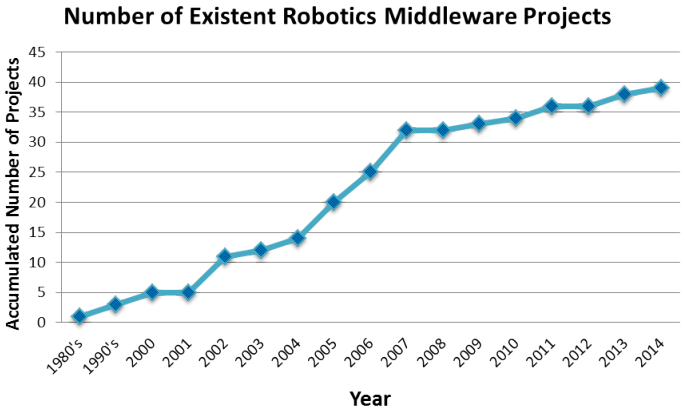


Figure 1. Accumulative number of existing robotics middleware projects by year.

more suitable for each part of the intended application, how they could be interfaced, and even benchmarking them. Therefore, we would like to contribute to the study of standards for highly integrated robotic systems.

Ideally, it should be possible to use any software element from any platform without restrictions, directly at runtime to avoid reinventing existing software elements (i.e. refactoring code), as if they were part of a single environment, where the administration methods for the heterogeneous system remain constant independently from the nature of its parts. Keeping a small set of abstractions as much as possible through normalization allows to generalize specific administration tasks. The number of possible paths for performing administration tasks on a heterogeneous system depends on various factors. When a single platform is used for creating a robotic system, its administration method relies on the set of abstractions provided by the development platform, the abstractions introduced by the developers for a specific application, which depend on the developer itself and on how customizable the platform is (e.g. if it provides a component model or not, as well as the model's manipulability and restrictiveness), and how restrictive their OS and/or hardware dependencies are. However, when two or more platforms become involved in the creation of a single robotic system (i.e. a highly integrated system), the number of abstractions increases, multiplying the amount of possible paths to take for a single administration method, making it daunting for the developers to work with the system. Various approaches for normalizing abstractions in different contexts and with different scopes are presented next.

**Robotics Middleware:**

Roughly speaking, middleware is a "glue" for joining a distributed set of software elements to build larger software applications. This allows the reuse, to some extent, of such software elements in different applications because they employ the same base abstractions. Formerly, middleware has been employed for various other purposes than robotics. Today, robotics middleware has become the modern approach for creating software in

the robotics community, enabling the collaborative development of complex systems and opening paths for new technology, currently existing more than 30 projects (Table 1 [1–9]). In the late 1990s open-source platforms began to appear (e.g. Player/Stage [10]). Later, emerging fields such as cloud robotics (e.g. Rapyuuta [11, 12] and RSi-Cloud [13] using RSNP [14]), started to adopt features from middleware for their development. Then, between 2007 and 2008 (Fig. 1), the number of middleware projects per year decreased, indicating a possible beginning of the consolidation of concepts in this field. Generally, since in robotics middleware a heterogeneous software system is built and used inside a single platform, the platform's abstractions are available natively only for the software elements designed under the middleware's policies. Due to the vast number of platforms with distinct aims, design policies and abstraction levels, there has been a discussion ([2], [9] and [15]) on the problems derived from the heterogeneity of highly integrated systems that prevent having a one-for-all solution or a de-facto standard in robotics middleware.

**Bridging Communications:**

To cope with the heterogeneity required in highly integrated systems, some middleware platforms are now providing communication bridges to support other platforms on their own. This is another level of normalization of abstractions that involves distinct middleware platforms, consisting on parsing and/or mapping abstractions between the participant platforms, and can be done through a software interface which may or may not use an intermediate communication protocol. This is the case of ROS-Bridge [16], which enables communication between ROS and software elements developed in another platform. Nevertheless, this is not a generalized solution, targeting and centering development in ROS. On the other hand, MARIE [17] proposed a set of common adaptors with basic managers to bridge communications among software elements in different platforms. However, to operate and monitor both software elements and utilized platforms, it is still complicated and tedious to deal with each of the internal platforms' administration methods and each possible task combination. Moreover, the integration needs to be done by specialists in this field, which is still prone to ad-hoc implementations. These projects provided valuable solutions that break communication barriers, but do not take into account the effect that bridging has for the developers on the overall usability when administering the highly integrated system.

**Formalized Component Models:**

A normalization of abstractions targeting a wide compatibility among platforms can be done during the early development stage of software elements through formalized component models. This way of normalizing abstractions encourages "best practices" for programming robotics software, making the developers to follow methodological patterns when developing software elements for different middleware platforms, allowing to port code among them. A clear separation of concerns is required to highlight the abstraction similarities among platforms; there being four main concerns (4Cs) identified by Radestock and Eisenbach [18]: computation, coordination, communication and configuration. Examples of formalized models are the GenoM [19] and BRICS [20] projects. GenoM proposes formal descriptions to generate a software element template independently from the targeted middleware platform through the GenoM3 tool, where it is also possible to apply the BIP methodology [21] for the development of real-time layered components. As for BRICS, the composition concern is introduced and the BRICS Component Model (BCM) was proposed, applying the four levels of abstraction defined by the Object Management Group (OMG) [22]; through the BRIDE tool, templates can also be generated to create software elements on the supported middleware platforms. Both projects offered good approaches for the cross-platform normalization issue. However, since this approach is targeted to early software development stages, it would imply to refactor the already available software el-

ements to make them work natively on a single platform, making it difficult to use the models in a direct runtime approach when a highly integrated system is required to be built from existing parts and off-the-shelf components.

### 1.1 *Research objective*

The objective of our research is to set the bases for switching the "common ground" of robotics software composition from a platform-driven level (i.e. relying on a particular platform that defines its own abstractions) to a model-driven level (i.e. relying on a model that defines normalized abstractions to be applied on any platform), focused on administering highly-integrated heterogeneous robotic systems. We identified from the previous approaches (middleware, bridging interfaces, and component models), that it is necessary to normalize at runtime the abstractions available in the existing platforms to compose and use highly-integrated systems without the need to refactor existing systems, thereby improving the usability in the administration of the employed set of software and development tools. To embody this normalization concept, as implied in [7] and [15], interoperability among platforms is a main requisite. This is achieved when data and services can be accepted from/provided to different parties, and when the means to manipulate such interactions are provided. Thus, the implementation of the proposed normalization method must comply with the following three requirements:

(1) Piping resources from one system to another: Bridged connections across platforms and a normalized system model can be combined to have common abstractions among platforms at runtime, minimizing the use of ad-hoc connection links.
(2) Executing basic commands in an integral way: Since there will be many specialized software subsystems that work properly on a certain platform, we need a way of executing basic commands on the participant platforms to issue services in a unified way.
(3) Operating the integral system: A front-end for manipulating the highly-integrated system's data and service interactions is provided (e.g. in a debugging task), where the user expects it to have higher usability than utilizing the development tools independently.

Therefore, we propose the Framework for Integration of Elements and Resources by Roles (FIERRo): it sets the paradigm of "Roles" to define a novel method of creating and reasoning a system model by normalizing abstractions of middleware-based software into elements and resources, where mappings among them are set to compose a heterogeneous system at runtime. Then, platform-agnostic models of a robotic software system are produced, allowing the developers to use the system straightforwardly at runtime through its implementation in a cross-platform infrastructure and a graphic user interface, which are introduced in the following section.

### 1.2 *General Overview*

Our proposals can be outlined through the previously introduced 5 concerns ([18] and [22]):

(1) Composition: Defines how the system is coupled and could be coupled to others.
   - The Framework for Integration of Elements and Resources by Roles (FIERRo) (Section 2) is employed to compose a highly-integrated robotic system through the HyperBot GUI (Section 3), where the system can be visualized and manipulated with the use of a diagram built from Roles.

(2) Communication: Defining how the results of computations are being communicated.
   - Along with each middleware's protocol, we employ our Intelligent Cross-Platform Interface (ICPI) (Section 3), the runtime cross-platform infrastructure that implements FIERRo for administering and reasoning the highly-integrated system.

(3) Configuration: Setting parameters to define element behaviours.
   - Done through middleware services and/or configuration files.

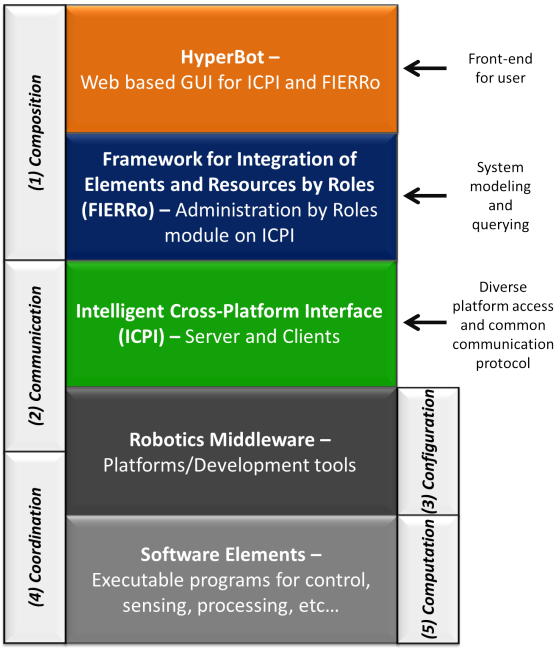


Figure 2. Layers of the proposal and their pertinent concerns.

(4) Coordination: Indicating when the elements must change their behaviour.
- Achieved through the subsystem's/software element's internal structure (e.g. state machines).

(5) Computation: Processing core functions.
- Done by the software elements inside the middleware platforms.

To summarize: in a system the software elements are in charge of the computation and coordination, the middleware platforms provide services for the communication and configuration while assisting the coordination, the ICPI augments the communication, and the FIERRo (through HyperBot) enables the composition (Fig. 2).

We employed two different modern middleware platforms to carry out a proof of concept and a usability test (Section 4). They were selected due to their relevance in the robotics community:

- Robot Operating System (ROS) [23]: Released in 2007 by Willow Garage, now maintained by the Open Source Robotics Foundation [24]; it actively promotes the collaborative development of robotics software.
- OpenRTM-aist [25]: Implements the RT-Middleware's (RTM) RT-Component (RTC) specification [26], established as standard in 2008 by OMG [27] and the Japanese AIST; it defines software design patterns and seeks for system reliability.

Our test implementations will be based on the integrated use of both platforms for specific tasks, performed by test subjects. Conclusions are discussed thereafter (Section 5).

## 2. Framework for Integration of Elements and Resources by Roles (FIERRo)

This framework is pertinent to the composition concern, as we recognize the importance of reusing elements from different platforms (composability), while keeping predictable behaviours once the behaviours of the other elements are known (compositionality). The general concept of our framework is to create a dynamic and flexible relational database holding the composition of the system. There are many "elements" that compose a robotic system during an application lifecycle, independently from the platform(s) used. In this context, an element is: a normalized

abstraction for any entity inside a middleware platform (and if necessary, its complementary tools). Elements can become "resources" when they can be resolved, accessed and retrieved; typically possible through an access point (e.g. a URL address). Then, a robotic system is composed of a set of elements (and resources) with explicit or implicit relationships among them. We are providing a means to allow the elements in charge of the other 4Cs to interact with each other, regardless of their nature.

## 2.1 *Composing Systems by Roles*

By definition, a "Role" is a customary function. The purpose of a Role in this context is to group a set of elements that compose an activity [28]. We define Roles as a "Subject-Verb-Object" (SVO) sentence (Fig. 3), a widely used word order [29]. Verbs are actions performed by an object (i.e. subject) over another (i.e. direct object). Therefore, the "object" and "action" entities are derived. Such entities are related to elements (and resources) composing them:

- Objects: Represent a thing in the system/environment (e.g. a motor, a ball, etc...) and are related to elements that represent data about them in various formats (e.g. data streams, static literal values, files, etc...).
- Actions: Represent performed processes (e.g. to move, to bounce, etc...) and are related to elements that represent runnable software, serving the robot to perform the intended action (e.g. executable files, firmware, scripts, etc...).

The scope of a Role can range from small subsystems comprised of a few pieces of software to bigger subsystems that involve the use of an entire platform for their exclusive execution. The use of Roles allows the composition of systems out of subsystems.

An element can have many representations and access points that could be scattered among the different objects and actions due to the heterogeneous nature of the system. For this, mappings between those representations and access points are made to achieve inter-element associations (e.g. associating data related both to an action and an object). As an example, let a Role be "GamePad-Drive-Motors" (Fig. 4). The "GamePad" object is related to the data set named COMMAND, which in turn has "AXIS_0" and "AXIS_1" data fields. Likewise, the "drive" action is related to the "GamePadX" software element, which has "Port_5" and "Port_8". Since the AXIS_0 and AXIS_1 data fields are intended to represent the same data as Port_5 and Port_8 correspondingly, a mapping between these element pairs can be created, indicating that AXIS_0 and AXIS_1 are synonyms of Port_5 and Port_8 respectively. Mappings are not limited to one-to-one relationships, nor to elements inside a Role, allowing mappings with elements that are part of other Roles. Elements and their mappings help in the normalization of abstractions.

### *2.1.1 Chaining Roles*

In a system, each subsystem is assigned to do a specific task (having its own dependencies and resources), while maintaining a link for interaction with the other subsystems that may or may not be distributed in separate platforms. Here as Roles allow the grouping of elements to form distinct subsystems regardless of their nature, a robotic system can be composed by chaining Roles together (Fig. 5). After chaining Roles and creating additional mappings that help to normalize abstractions, a model describing the composition of the software system is created and can be represented in a diagram.

## 2.2 *Serialization*

To store and retrieve the system composition made by using Roles, a way to serialize this information is needed. For encoding Roles, objects, actions, mappings and other elements, we selected the Resource Description Framework (RDF), a standard for encoding metadata and other knowledge on the Semantic Web [30] which is well-suited to our framework's general idea.

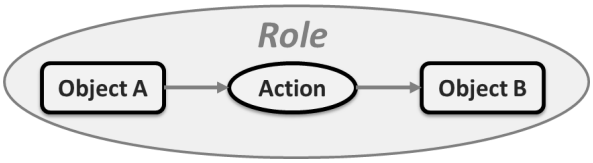


Figure 3. A Role definition.

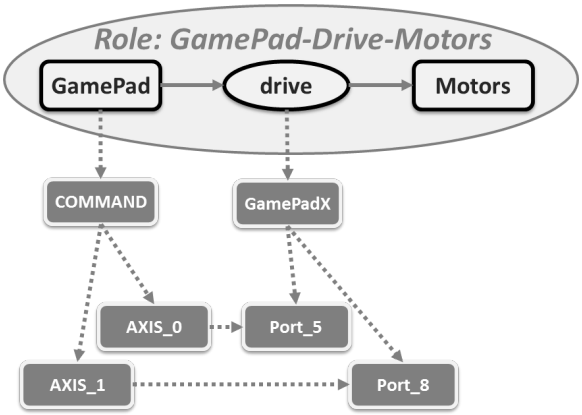


Figure 4. Example Role: Gamepad-Drive-Motors.

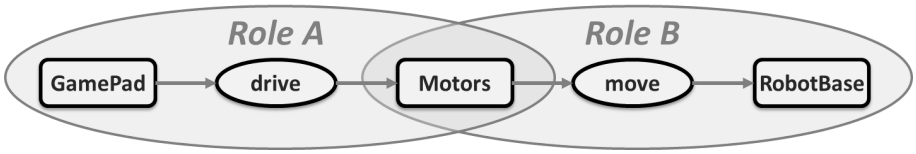


Figure 5. Chaining Roles.

This structured information can be spread in a distributed and decentralized manner. Items are stated in the form of Uniform Resource Identifiers (URI); for instance, every Role, object, action and element in our framework can be represented as a URI. Through RDF, it is possible to define elements and state their type by using the "rdf:type" predicate arc. Custom predicate arcs (i.e. properties) can be declared to define ontologies. For this FIERRo implementation, the "fierro" namespace was defined with the initial possible properties (and types) described in Table 2. For formatting, the RDF/XML specification was used. An example of an RDF definition for a Role is shown in the following listing (other elements can be defined similarly):

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fierro="http://www-robot.mes.titech.ac.jp/fierro#" >
  <rdf:Description
  rdf:about="http://www-robot.mes.titech.ac.jp/MYSUBJECTmyverbMYOBJECT">
    <rdf:type rdf:resource="http://www-robot.mes.titech.ac.jp/fierro#role"/>
    <fierro:ID>MYSUBJECTmyverbMYOBJECT</fierro:ID>
    <fierro:subject rdf:resource="http://www-robot.mes.titech.ac.jp/mysubject"/>
    <fierro:action rdf:resource="http://www-robot.mes.titech.ac.jp/myverb"/>
    <fierro:directObject rdf:resource="http://www-robot.mes.titech.ac.jp/myobject"/>
  </rdf:Description>
</rdf:RDF>
```

Table 2. FIERRo namespace properties (and types)

| Item | Type | Properties (from Type col.) |
|---|---|---|
| Role | role (a) | (b), (c), (d), (e), (q) |
| Object (Subject & Direct Object) | object (b), subject (c), directObject (d) | (e), (f), (i), (q) |
| Action | action (e) | (b), (d), (k), (f), (i), (q) |
| Data | data (f) | (g), (q) |
| Structured (or Hierarchical) Data | structuredData (g) | (h), (q) |
| Structured Data Field | field (h) | (q) |
| Service | service (i) | (j), (q) |
| Operation | operation (j) | (q) |
| Software | software (k) | (l), (q) |
| Software Element | softwareElement (l) | (m), (n), (o), (p), (q) |
| File | file (m) | (q) |
| Executable file, script, firmware | process (n) | (q) |
| Function or method | function (o) | (p), (q) |
| Data Port or Argument | port (p) | (q) |
| Mapping | mapping (q) | any of the above |

## 2.3 *Administration by Roles method*

To enable the administration of the defined Roles (thus the system in general, including the participating software elements and platforms), a reasoning mechanism is required. This method serves for that purpose; it has been revised several times during the course of our research. While there can be various types of queries to the system (e.g. getting and setting data, piping resources, initializing and executing software) the common step for them is to resolve the pertinent elements in the system (i.e. checking for their existence and location) to complete the query. This is discussed in the next subsection (for implementation, see section 3.2.2).

### *2.3.1 Resolution of elements*

Elements (and resources) in the system can be resolved through propositional calculus. Our framework defines a set of base premises for reasoning the robotic system's composition. As for the Role level, the base premises are:

$$\forall \text{ R, O, A; where: R:= a Role, O:= an Object, A:= an Action}$$

$$(R \rightarrow O_i) \wedge (R \rightarrow A) \wedge (R \rightarrow O_{i+1}) \tag{1}$$

$$O_i \rightarrow A \tag{2}$$

$$A \rightarrow O_{i+1} \tag{3}$$

These three initial premises state that there must be one action and two objects for a Role to exist, also that the existence of a first object implies an action, and that an action implies the existence of a second object. Each object and action are associated with elements that describe them, which in turn can be associated with other elements. Therefore, the following premises can be used:

$$\forall \text{ O, A, E; where: O:= an Object, A:= an Action, E:= an Element}$$

$$O \rightarrow E \tag{4}$$

$$A \rightarrow E \tag{5}$$

$$E_j \rightarrow E_k \tag{6}$$

The three premises shown above state that for an element to be declared, there must be an object, an action or another element already declared. For instance, when mappings are created between elements, premise (6) applies.

Taking the previous example of the Role "GamePad-Drive-Motors" (Fig. 4), we may want to retrieve the data represented by the "AXIS_0" element. For this, the query engine needs to know the resources: which (data) port is to be accessed, which running process contains this port (e.g. a software element), and where the process is hosted (the host is represented by a Role; the hosting mechanism is summarized in section 3.2.2). First, the "Port_5" and the "GamePadX" elements can be resolved by knowing there is an "AXIS_0" element:

$$AXIS\_0 : Starting\ Premise \tag{7}$$

$$AXIS\_0 \rightarrow Port\_5 : Premise(6) \tag{8}$$

$$GamePadX \rightarrow Port\_5 : Premise(6) \tag{9}$$

$$Port\_5 : Modus\ Ponendo\ Ponens(7, 8) \tag{10}$$

$$GamePadX : Modus\ Ponendo\ Ponens(9, 10) \tag{11}$$

Then, to know where the "GamePadX" process is hosted, the "GamePad-Drive-Motors" Role can be resolved as follows:

$$GamePadX : Starting\ Premise \tag{12}$$

$$Drive \rightarrow GamePadX : Premise(5) \tag{13}$$

$$GamePad\text{-}Drive\text{-}Motors \rightarrow Drive : Premise(1) \tag{14}$$

$$Drive : Modus\ Ponendo\ Ponens(12, 13) \tag{15}$$

$$GamePad\text{-}Drive\text{-}Motors : Modus\ Ponendo\ Ponens(14, 15) \tag{16}$$

Finally, it can be concluded that the "Port_5" exists and can be accessed by making a call to the "GamePadX" process through the "GamePad-Drive-Motors" Role, which is the host. A similar procedure is followed for other types of queries.

As for the implementation of this method, in a serialized Role an RDF statement can be seen as a premise, which can be used to make inferences from other stated premises as described above. For this, we suggest the use of an SPARQL engine to automate this process, where SPARQL is a semantic query language for RDF databases [31]. The implementation of this into the ICPI is discussed in the following section.

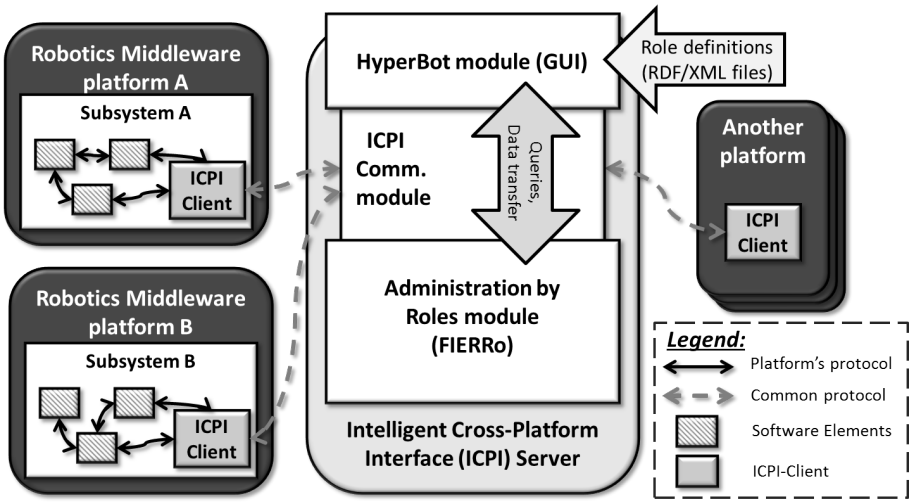


Figure 6. The Intelligent Cross-Platform Interface. The Role definitions are input through the HyperBot GUI, which communicates with the Administration by Roles module (implementing FIERRo) for processing them, as well as querying the system and transferring data requested by the user. The Administration by Roles module uses the ICPI Comm. module to access the clients to complete queries when required.

## 3. Intelligent Cross-Platform Interface (ICPI)

This interface deals to some extent with the communication concern by providing a common communication protocol when required by the instantiated subsystems. The ICPI provides the functional infrastructure to our framework and is based on the client-server approach [32]. This interface enables the access of resources contained in the software elements that are hosted by different platforms (e.g. middleware platforms) as in Fig 6. Since we are already dealing with heterogeneous software and different platforms that may have their own dependencies/installation requirements, for the ICPI we intend to have as much as possible a flexible and installation free environment, in a way that the users can manipulate their systems on-the-fly. Technologies that adapted well to this concept such as Java and HTML5 were used for this implementation. The main functional parts, the ICPI-Clients and the ICPI-Server, are explained next.

### 3.1 *ICPI-Client*

The client is a software program able to perform basic administration tasks within a platform. The way such tasks are implemented may vary among platforms, and depending on their imposed restrictions, the client may be hosted inside or outside of them. The client provides a way to perform a given task regardless of the platform, like sending data to/getting data from software elements, as well as calling functions to operate on them by using the corresponding protocols. For example, considering RTM and ROS, a command for connecting to a Data Port in RTM should be similar to a command for subscribing to a ROS Topic. Additionally, connecting Data Ports from RTM to Topics in ROS should be seamless. The ICPI-Client acts like a "gate" that allows one subsystem to interact with other different subsystems, supported by the ICPI-Server.

### 3.2 *ICPI-Server*

The server is a platform independent software program that handles the queries in the system and executes commands with the help of the ICPI-Clients. It provides the communication means for the clients (hence the subsystems hosted by the various platforms), as well as modules for managing them. It is composed of three main modules: The ICPI Communication module, the Administration by Roles module and the HyperBot module.

### *3.2.1 ICPI Communication module*

This module is in charge of setting a common communication protocol among the subsystems. Websockets were selected as the base protocol for communication (compatible with HTML5, see section 3.2.3). Over this protocol, the ICPI has its own protocol for queries. The latest version of the protocol is being used here. It is defined as the following concatenated character string:

$$\begin{aligned} ICPI\text{-}QueryMessage = Destination + \text{“!}sn\text{!”} + Sender + \text{“!}id\text{!”} + MessageID + ... \\ ... + \text{“!}ak\text{!”} + Acknowledgement + \text{“!}fx\text{!”} + FunctionName + \text{“!}ar\text{!”} + Arguments + ... \\ ... + \text{“!}br\text{!”} + BroadcastFlag \end{aligned} \quad (17)$$

Where,

***ICPI-QueryMessage:*** Resulting string.
***Destination:*** Name of the ICPI-Server/Client that is being queried.
***Sender:*** Name of the ICPI-Server/Client that is issuing the query.
***MessageID:*** Optional identifier that can be used to set priorities or distinguish between similar messages.
***Acknowledgement:*** Flag defining if the query requires a reply (0 = no, 1 = yes).
***FunctionName:*** Name of the queried function.
***Arguments:*** List of literal and/or numerical values; each one separated by the following concatenated string: *“(” + Index + “)”*, where *Index* is an integer starting from 0.
***BroadcastFlag:*** Flag defining if the message must be broadcasted to other ICPI-Clients (0 = no, 1 = yes).

An example of *ICPI-QueryMessage* is shown next:

$$\text{“}icpicClient0!sn!icpicClient1!id!0!ak!0!fx!MyFunction!ar!(0)0.0(1)1.2(2)true!br!0\text{”} \quad (18)$$

The ICPI-Server includes a Websocket Server that is accessed by the ICPI-Clients. Queries from other clients or modules can be made through this protocol to the ICPI-Server and vice-versa. The following two modules make use of this module for communication.

### *3.2.2 Administration by Roles module*

The ICPI-Server comes with a module that enables the administration of the system through Roles (implementing the method of section 2.3) with the support of a SPARQL engine to resolve elements and resources. When the robotic system has been composed using FIERRo, the serialized system is processed and stored in the internal RDF database. Then, several high-level orchestration operations can be automated through this module and the ICPI-Clients (a detailed example is shown in section 4.2, Fig. 12). The module's flow diagram is summarized in Fig. 7.

### *3.2.3 Hyper-High Level Interface for Service Robots (HyperBot) module*

HyperBot is a Graphic User Interface designed to work with the ICPI-Server (hosted in the internal HTTP server). This module serves as a visual front-end for our framework, having direct communication with the ICPI-Server [33]. It has a screen called “System Developer Screen”, where RDF/XML files for Roles, objects, actions and other elements can be created/loaded by using a set of buttons and inputting the required information. The ICPI-Server processes the files and the composed system is displayed here in the form of a diagram (example in section 4.1, Fig. 9). Then, the user is able to make queries in the composed system for different administration purposes, for instance: getting and setting data, initializing software elements, calling functions, and automating middleware connections. Section 4.2 presents a detailed example of this through a usability test. Following the idea of an installation free environment, HTML5 was

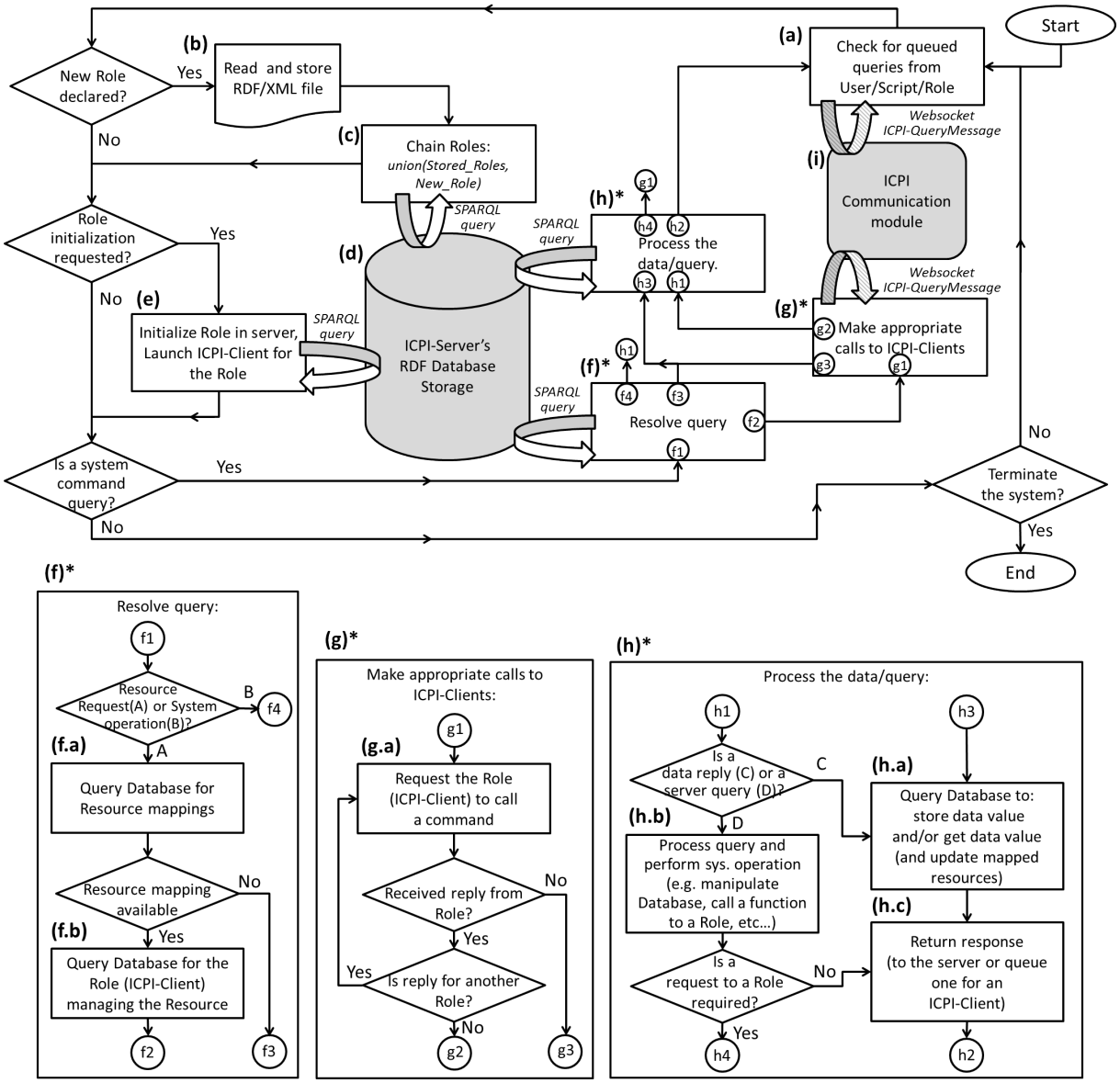


Figure 7. Summarized flow-chart of the Administration by Roles module. In (a) the ICPI-Server checks for queued queries coming from Hyperbot (issued by the users or scripts), from internal server scripts, or from a Role (ICPI-Client) by using the ICPI Communication module in (i). In (b) the serialized Roles are read and stored, and in (c) they are chained together with the previously stored ones, updating the database in (d). Upon the initialization of a Role as in (e), its execution script is automatically generated and initialized in the server, then the Role is assigned an ICPI-Client, which its configuration is queried in the database, and finally it is launched. In (f), incoming queries are resolved into resource requests or system operations; this block is detailed in (f)*, where (f.a) and (f.b) use the SPARQL engine. In (g), command calls to the pertinent Roles (ICPI-Clients) are made; in the detail shown in (g)*, (g.a) uses (i) to access the Role. Then in (h), data replies or server queries coming either from (f) or (g) are processed, where (h.a) and (h.b.) use the SPARQL engine, then (h.c) gathers the result of the operation and returns it to the pertinent party. The SPARQL queries shown in the interaction with the RDF Database can refer to specific database operations (e.g. data insertion, deletion, etc...) and/or a reasoning with the previously stated premises (section 2.3.1). As for the interactions with the ICPI-Clients through (i), ICPI-QueryMessages are sent to/received from the ICPI-Clients representing Roles or acting as a GUI (as HyperBot).

chosen for HyperBot's development, enabling the usage of this GUI on the major web browsers and operating systems (coded in pure JavaScript and CSS to avoid any plug-in installation). HyperBot is scalable, where other screens can be added to enhance its usage, such as screens for designing system operation panels, for scripting administration tasks, and for providing other ways of visualizing the system (e.g. 3D views).

Table 3. Hardware and Software setting for the Proof-of-concept. Note that "exe file" refers to executable files in Windows 7, "sln file" refers to a Visual Studio 2008 (VS2008) solution file, "sh file" refers to an script file for Ubuntu 12.04's Terminal, and "ttt file" refers to a V-REP scene file.

| **Hardware** | **Software** | | |
|---|---|---|---|
| Devices | Files | Data Interfaces | Function Interfaces |
| Gamepad | GamePadX RTC (exe file) | GamePadX RTC data ports | V-REP's ROS node services |
| WLAN router | GamePadX RTC (sln file) | HeliosIXControl RTC data ports | |
| HELIOS IX robot | HeliosIXControl RTC (exe file) | V-REP's ROS services' arguments | |
| Robot's Laptop PC (Win. 7, Firefox) | HeliosIXControl RTC (sln file) | | |
| Desktop PC (Ubuntu 12.04, Firefox) | V-REP w/ ROS Node (sh file) | | |
| Tablet PC (Android 4.4, Chrome) | Robot Scene file (ttt file) | | |

## 4. Implementation and Usability validation

To evaluate our proposal, we designed various tests based on a series of tasks for composing and administering a robotic system that uses two different middleware platforms, RTM and ROS. First in section 4.1, we conduct a case-study of normalization to make a proof of concept and validate our proposal. Then as stated in the research objective, there are three requirements for interoperability, which make possible to embody the concept of normalization of abstractions at runtime. Therefore, three tasks representing each of those requirements will be used to validate the usability of our framework; they are:

(1) Piping resources from one system to another, regardless of their host platform.
(2) Executing basic commands in an integral way, regardless of the utilized platforms.
(3) Operating the integral system, regardless of its heterogeneity.

Therefore, we consider that by meeting these requirements, we can enable the administration of a heterogeneous system as discussed before. The performed tests are described in section 4.2, then results are shown in section 4.3 followed by a discussion in section 4.4.

### 4.1 *Proof of concept*

To validate the functionality of our proposal, we composed a heterogeneous system with the HELIOS IX mobile robot [34] as a case-study. It contains the items listed in Table 3, and in Fig. 8 an overview of the setting is shown. As for the implementation of our approach, two items were required: The ICPI-Server (which provides the ICPI-Clients through a download) and a web browser to access the HyperBot. The ICPI-Server was instantiated on the robot's Laptop PC for convenience. The system is complex enough to be normalized, and involves the use of the requirements previously mentioned. The gamepad is used to drive the crawlers of the robot, then the robot provides feedback on its status, which is displayed in the V-REP robot simulator [35]. The programs that deal with the gamepad and the robot's control are hosted on one middleware platform (RTM) and running on the robot's Laptop PC, while the simulator program is hosted on another platform (ROS) running on the Desktop PC. Next, the system was composed through the HyperBot GUI by using the Tablet PC. The robotic system was composed successfully by using three Roles: "Gamepad-Drive-Motors", "Motors-Move-RobotBase" and "Simulator-Display-RobotBase" (Fig. 9). After the system was composed, three ICPI-Clients were launched by the ICPI-Server (in the pertinent machines), each one representing one Role. Then, the Roles were initialized by clicking on their respective nodes in the diagram of the HyperBot's System Developer Screen, automatically connecting the required RTC ports, and piping data from those ports to the corresponding ROS service's arguments. The subsystems interacted with each other as expected; the gamepad was used to drive the robot on a sample path, and was visualized through the V-REP simulator. As an additional exercise, the VS2008 solutions were opened directly from HyperBot to edit the code.

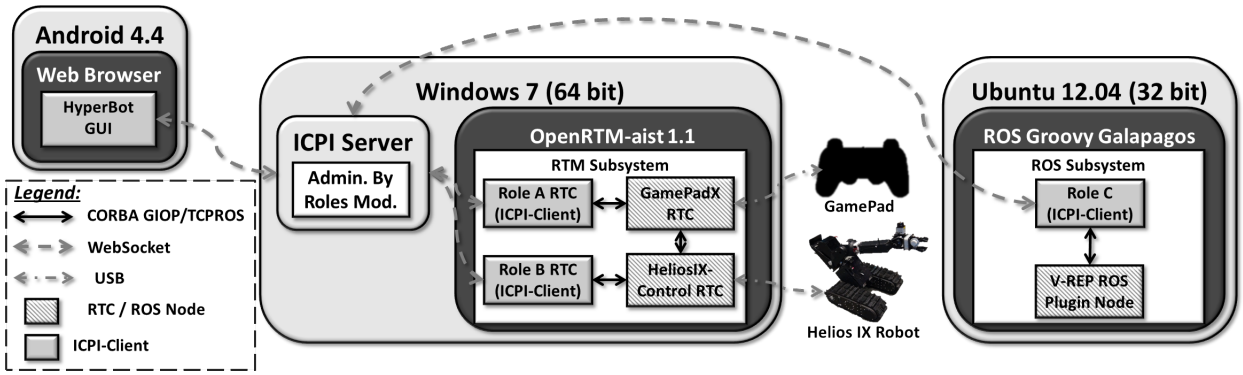


Figure 8. Overview of the system setup for the Proof-of-concept. See Fig. 6 for details on the ICPI-Server, Fig. 7 for details on the Admin. by Roles module, and section 3.2.3 for details on the HyperBot GUI. Role A = "GamePad-Drive-Motors"; Role B = "Motors-Move-RobotBase"; Role C = "Simulation-Display-RobotBase".

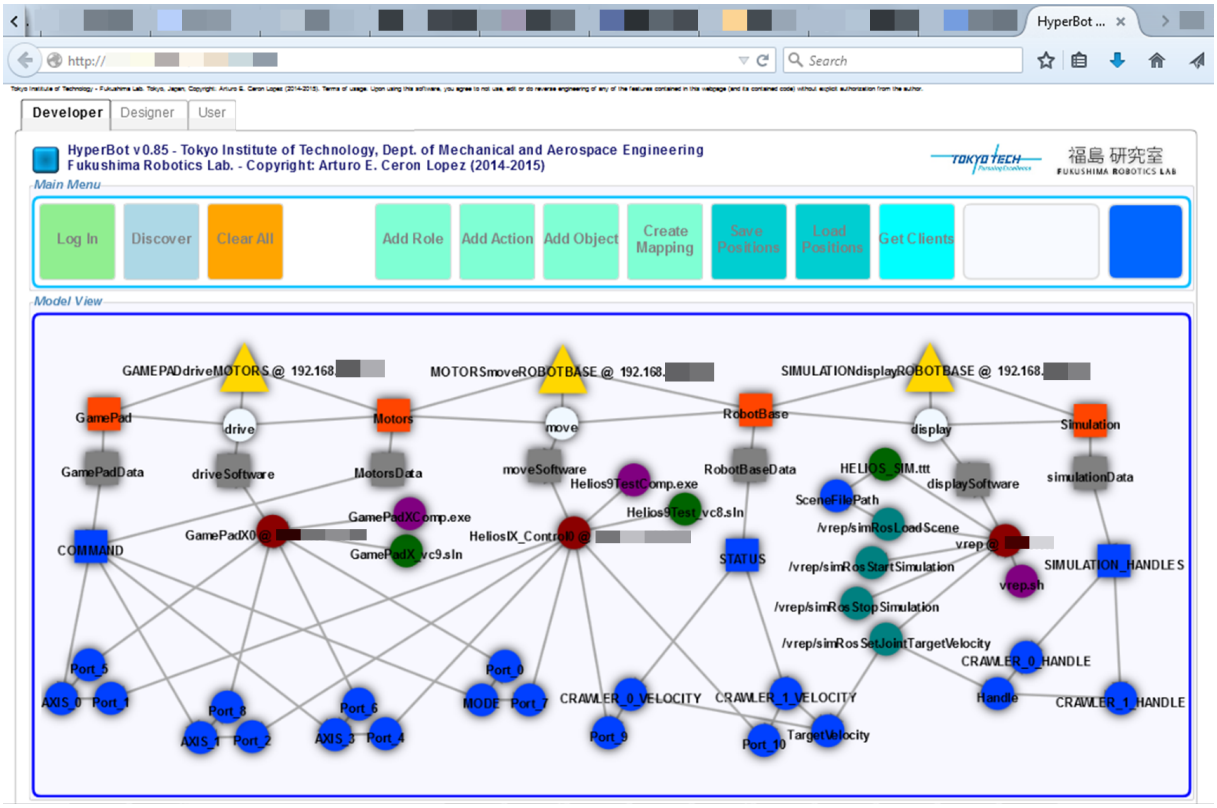


Figure 9. HyperBot GUI displaying the Role diagram of the Proof-of-concept.

## 4.2 *Usability test*

The usability test will give us an idea of the extent to which our proposal improves the overall usability of the task of composing and administering robotic systems when using separate development tools in conjunction. A way to get an estimate of the usability of our proposal is by having the users grade it on a scale and by measuring their task times. The measured parameters were the following:

- System Usability Scale (SUS) Score: The subjects completed the SUS questionnaire [36] (ten-item form) for both approaches (being a within-subjects design test, i.e. paired samples of SUS scores from approaches A and B). SUS is a simple scale giving a global view of subjective assessment of usability; its score range is 0–100. This score can be decomposed

Table 4. Main software tools for Usability test.

| Approach | Software Tools |
|---|---|
| (A) Benchmark | RT-Middleware: OpenRTP (RTC-Builder and RT-System Editor in Eclipse), RT-naming-service.<br>ROS: catkin_init_workspace, catkin_make, roscore, rosservice, rostopic and related commands in Command-line Terminal.<br>Simple text editor (e.g. Notepad). |
| (B) Proposed | Web browser (HTML5 compatible).<br>ICPI-Server (with HyperBot GUI, Administration by Roles module and ICPI-Clients).<br>Simple text editor (e.g. Notepad). |

into two factors for further analysis: usability alone and learnability [37]. Our expectations were that the scores for approach B would be higher, meaning better usability. The internal reliability of the SUS score samples can be evaluated by the Cronbach's alpha [38], giving an estimate of the correlation among test items measuring a construct.

- Time: The time required by the subjects to complete the tasks in each approach was measured. We expected that the completion of tasks in approach B would take less time. Additionally, by using the Novice-Expert (NE) ratio [39] we roughly compared the amount of time required between the two subject groups on each approach:

$$NE = \frac{NoviceTime}{ExpertTime} \tag{19}$$

We tested the procedure for normalizing abstractions at runtime and the three administration tasks of interest with a sample size of 12 subjects (4 different institutions and 5 nationalities) as recommended in [37] and [40]. We selected the subjects according to two profiles, each one forming a group of 6 subjects:

- Expert group: The subjects are directly involved in robotics projects where middleware is actively used.
- Novice group: The specialization of the subjects is either mechanical or software engineering. Their programming skills are more suited to those fields than robotics itself, but are interested in robotics topics.

Except for our procedure of normalizing abstractions at runtime (section 4.2.1), which is to the best of our knowledge unavailable in the previous approaches, the tests consisted of a series of tasks to complete by using two different approaches:

- Approach A. Benchmark: Use of the tools provided by the middleware platforms and operating systems.
- Approach B. Proposed: Use of FIERRo through the ICPI and the HyperBot GUI.

From here on, the benchmark and proposed approaches will be referred to as approach A and B respectively. The tools used for each approach are listed in Table 4. For each approach, both groups were given a 25-35 minute tutorial on the tools needed for completing the required tasks.

The usability tests were based on the system from the proof of concept, with minor changes to keep the experiment simple and let the subjects compose and operate it in a reasonable time. The resulting setting is listed in Table 5, and in Fig. 10 the data flows and command paths can be visualized, marked in the following subsections as "DF$x$" and "CP$y$" respectively, where $x$ and $y$ are index numbers. The Roles for composing the system were: "Console-Drive-RobotBase", "Console-Show-RobotStatus" and "Simulator-Display-RobotBase" (Fig. 11). Here, the ConsoleIn RTC sets the angular speed of the robot's crawlers to the V-REP ROS node, and the operation's returned value is displayed in the ConsoleOut RTC. An example implementation of this operation in the Administration by Roles module (section 3.2.2) showing actual data flows is presented in Fig. 12. This was a supervised test; some subjects required slight assistance from the staff when asserting their actions while using the templates in both approaches, mainly

Table 5. Hardware and Software setting for the Usability test. Note that "exe file" refers to executable files in Windows 7, "sln file" refers to a Visual Studio 2008 (VS2008) solution file, "sh file" refers to an script file for Ubuntu 12.04's Terminal, and "ttt file" refers to a V-REP scene file.

| **Hardware** | **Software** | | |
|---|---|---|---|
| Devices | Files | Data Interfaces | Function Interfaces |
| Laptop PC (Win. 7, Firefox) with Virtual Machine (Ubuntu 12.04, Firefox) | ConsoleIn RTC (exe file)<br>ConsoleIn RTC (sln file)<br>ConsoleOut RTC (exe file)<br>ConsoleOut RTC (sln file)<br>V-REP w/ ROS Node (sh file)<br>Robot Scene file (ttt file) | ConsoleIn RTC data ports<br>ConsoleOut RTC data ports<br>V-REP's ROS services' arguments | V-REP's ROS node services |

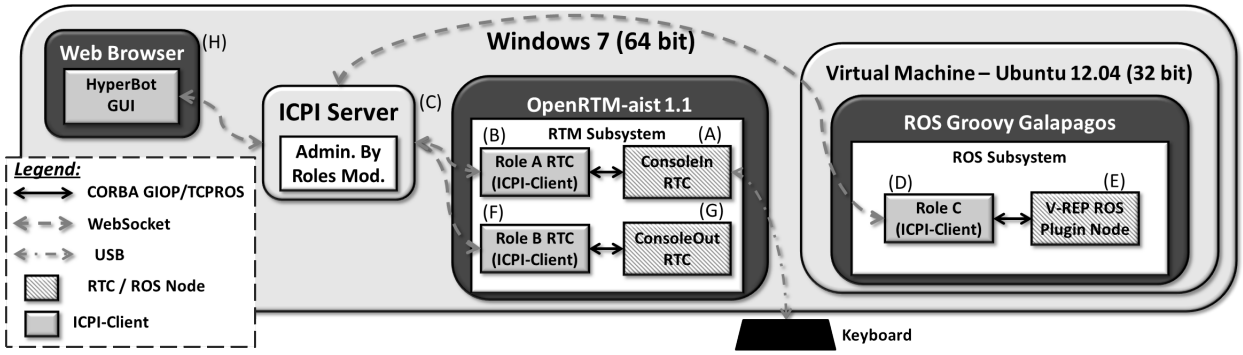


Figure 10. Overview of the system setup for the Usability test. See Fig. 6 for details on the ICPI-Server, Fig. 7 for details on the Admin. by Roles module, and section 3.2.3 for details on the HyperBot GUI. Role A = "Console-Drive-RobotBase"; Role B = "Console-Show-RobotStatus"; Role C = "Simulation-Display-RobotBase". Note *I for task 1 approach B: (DF1) = (A)→(B)→(C)→(D)→(E); (DF2) = (E)→(D)→(C)→(F)→(G). Note *II for task 2 approach B: (CP1) = (H)→(C)→(B)→(A); (CP2) = (H)→(C)→(D)→(E). Note *III for task 3 approach B: (DF3) = (A)→(B)→(C)→(H); (DF4) = (H)→(C)→(D)→(E); (CP3) = (H)→(C)→(B).

Table 6. Results for Normalizing abstractions and relating them at runtime, regardless of their nature. The analysis of variance (One-way ANOVA) for the time data of both groups laid a p-value of 0.458 (alpha = 0.05), meaning that there are no statistically significant differences between group means. Additionally, a NE ratio of 1.13 was calculated. Thus, we are inclined to conclude that novices would take approximately the same amount of time than experts for this task.

| Group | Time mean [min.] | Time std. dev. |
|---|---|---|
| Expert | 45.6 | 10.8 |
| Novice | 51.7 | 16.3 |

for approach A. Additionally, some subjects made minor mistakes while doing the tasks, such as misspelling items, considered irrelevant to the idea we want to validate. At the end, the results from the questionnaires and time measurements were collected and analyzed.

### *4.2.1 Normalizing abstractions and relating them at runtime, regardless of their nature*

In this test, the subjects normalized abstractions at runtime (e.g. RTC data ports and ROS Topics) into elements and resources (with their mappings). They were given RDF templates (XML formatted files) for Roles, actions, objects and mappings (similar as the listing in section 2.2); by using the keyboard and text editing software, they filled the templates with the necessary information. First, the Roles templates were filled, declaring the respective objects and actions. Next, for each object and action, the corresponding elements were declared (e.g. data fields and software elements) inside the templates. Finally, the normalization was completed by relating synonym elements and resources through mappings. The resulting files contained the description of the composed heterogeneous robotic system, where its diagram representation was shown on the HyperBot GUI (Fig. 11). As we consider that our proposal is relatively easy to use, we expected that both experts and novices would take nearly the same time to complete this task. The measured task times suggested that our expectations were correct (Tab. 6, Fig. 13).

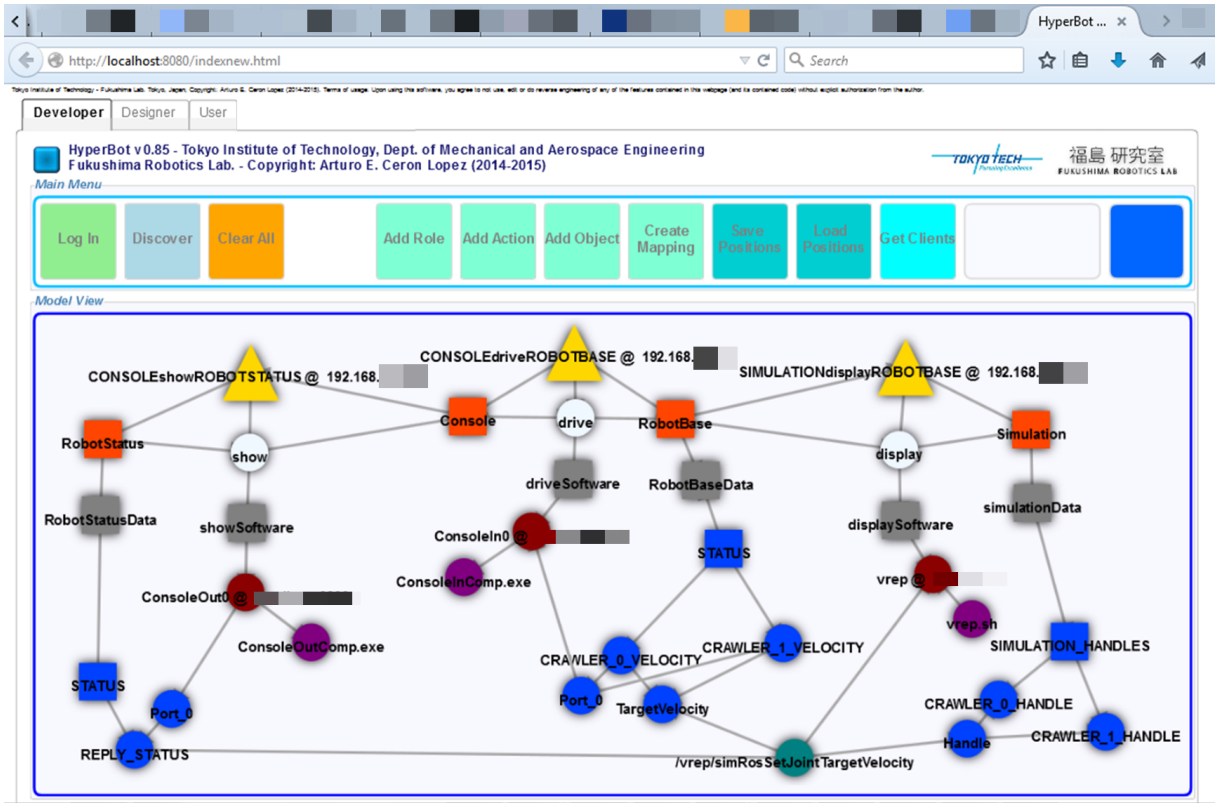


Figure 11. HyperBot GUI displaying the Role diagram of the Usability test.

### *4.2.2 Task 1: Piping resources across systems, regardless of their host platform*

This task's objective is to send data from a software element in a middleware platform to another element inside another platform (i.e. bridging data between an RTC and a ROS node). The assigned task was the following: The subjects entered an integer value corresponding to an angular speed for the robot's crawlers on the ConsoleIn RTC. Later, this value was sent to the V-REP ROS node to move the simulated robot forward/backward (DF1). After the operation's completion, the returned value of the function call was sent to the ConsoleOut RTC (DF2), displaying the value on the screen (see Fig. 10, note *I). The subjects proceeded as follows:

- Approach A: The subjects needed to write/compile code for creating a data bridge between RTM and ROS by implementing a standard TCP/IP client-server. The TCP/IP client-server was adopted as a benchmark because it is one of the solutions with the least dependencies. ROS-Bridge was not taken into account as it would only fit in this particular case-study using ROS, and we needed to make the comparison with a platform-agnostic interface that could be used in more general cases. The TCP client was hosted in the created RTC and the TCP server in the created ROS node.
- Approach B: The subjects generated a data bridge between RTM and ROS by loading and starting the previously composed robotic system in the HyperBot GUI, which transferred the RDF files to the ICPI-Server (for resolving the system) and made queries for automatically generating the required data bridges with help from the ICPI-Clients (as declared by the system model's mappings). The data was transferred from one party to another.

The subjects completed the corresponding tasks, being able to pipe information across the systems. According to the subjects' comments, this was the most troublesome task if using approach A, which agreed with our expectations. The results (Tab. 7) suggest that independently from the users being novices or experts, approach A has less usability.

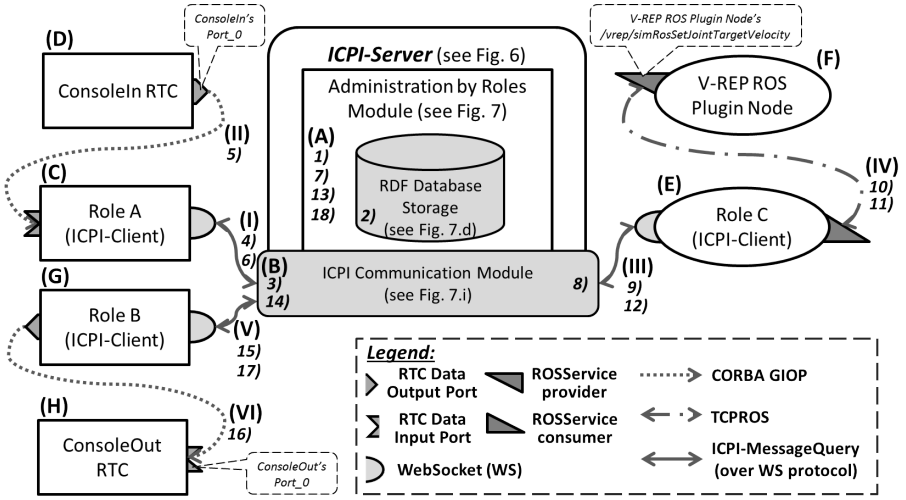


Figure 12. Data flows between middleware platforms for case-study. See Fig. 10 for details on the system setup for this case-study (Usability test), and Fig. 11 for the Role diagram representing this case-study. The following is an example implementation of a call to the function "/vrep/simRosSetJointTargetVelocity" as in Fig. 11 by using resource requests: *1)* As this function requires of the "Handle" and "TargetVelocity" arguments, a script was generated in (A) (see Fig. 7 (e)) to get their resource values before calling this function; *2)* Resources for "Handle" are requested (see Fig. 7 (f)), requiring only a database query (see Fig. 7 (h.a)); *3)* Resources for "TargetVelocity" are requested (see Fig. 7 (f)), requiring a call to an ICPI-Client (see Fig. 7 (g)) by using (B); *4)* An ICPI-MessageQuery through connection (I) travels to (C) requesting the data from "Port_0" in (D); *5)* (C) instantiates an RTC Data Input Port to read (D)'s "Port_0" using connection (II); *6)* (C) replies to (B) with the data from (D)'s "Port_0" through conn. (I); *7)* (A) processes this information (see Fig. 7 (h)); *8)* Now the server is ready to call the function (see Fig. 7 (f), Fig. 7 (h.b) and Fig. 7 (g)); *9)* An ICPI-MessageQuery through conn. (III) travels to (E) requesting a call to the function in (F); *10)* (E) parses the received request and calls the function using conn. (IV); *11)* The function is executed in (F), sending a reply back through conn. (IV); *12)* (E) receives the reply, which is parsed into a ICPI-MessageQuery and sent back to (B) through conn. (III); *13)* (A) processes the information (see Fig. 7 (h.a)), updating "REPLY_STATUS"; *14)* Since (H)'s "Port_0" is mapped with "REPLY_STATUS" (see Fig. 11), a data write request to (H)'s "Port_0" is issued (see Fig. 7 (f)), requiring a call to an ICPI-Client (see Fig. 7 (g)) by using (B); *15)* An ICPI-MessageQuery through conn. (V) travels to (G) requesting writing data to "Port_0" in (H); *16)* (G) instantiates an RTC Data Output Port to write data to (H)'s "Port_0" using conn. (VI); *17)* (G) replies to (B) if the operation was successful or not through conn. (V); *18)* Finally, (A) processes this information as in Fig. 7 (h), ending the scripted operation. Other data flows and command paths as the ones of Fig. 10 (DF$x$ and CP$y$) can be performed similarly.

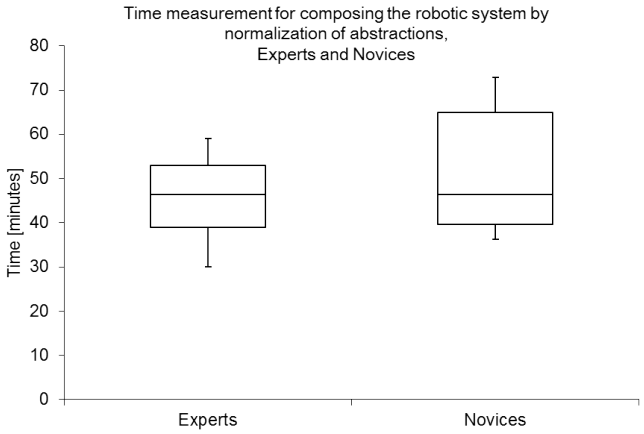


Figure 13. Time comparison between experts and novices for composing the robotic system by normalization of abstractions.

Table 7. Results for task 1: Piping resources across systems, regardless of their host platform. The obtained NE ratio for approach A (1.27) indicates that this task was slightly more difficult for novices than for experts. For approach B, The NE ratio was surprisingly 1.04, meaning that novices performed as well as experts.

| Approach | SUS mean | SUS std. dev. | Usability | Learnability | Cronbach's alpha | Time mean [min.] | NE ratio |
|---|---|---|---|---|---|---|---|
| (A) Benchmark | 18.6 | 12.7 | 20.2 | 12.5 | 0.74 | 49.2 | 1.27 |
| (B) Proposed | 72.1 | 16.7 | 74.7 | 61.5 | 0.88 | 6.80 | 1.04 |

Table 8. Results for task 2: Executing basic commands in an integral way, regardless of the utilized platforms. This task had the highest SUS scores (31.5 and 79.5), but also the highest NE ratios (1.48 and 1.33), suggesting that the experts are quite used to operate multiple applications simultaneously and are familiar to more than one operating system.

| Approach | SUS mean | SUS std. dev. | Usability | Learnability | Cronbach's alpha | Time mean [min.] | NE ratio |
|---|---|---|---|---|---|---|---|
| (A) Benchmark | 31.5 | 12.0 | 31.3 | 32.3 | 0.69 | 5.64 | 1.48 |
| (B) Proposed | 79.5 | 17.5 | 79.2 | 80.7 | 0.91 | 1.17 | 1.33 |

#### *4.2.3 Task 2: Executing basic commands in an integral way, regardless of the utilized platforms*

The objective for this task is sending commands to the executed software elements hosted on the independent middleware platforms. The assigned task was: To issue the "Activate" command to the CommandIn RTC in RTM (CP1), and to call the "/vrep/simRosStartSimulation" service at the V-REP ROS node in ROS (CP2). Then both programs reacted to the commands, executing them (see Fig. 10, note *II). The subjects proceeded as follows:

- Approach A: The subjects accessed both platform's tools. The RT-System Editor was used to "Activate" the said RTC, and the Terminal to call the ROS service through the "rosservice" command. Thereafter, the RTC changed its state to "Active" and the V-REP started the simulation.
- Approach B: The subjects accessed the HyperBot GUI. Through the Role diagram in the System Developer Screen, the subjects directly clicked on the required elements representing the CommandIn RTC and the V-REP ROS node, where from a pop-up menu the said commands were issued. Similarly, the RTC changed to "Active" and the V-REP started the simulation.

The subjects executed the commands successfully and the programs behaved as expected.The subjects commented that this was the simplest task, confirmed by the high SUS scores (Tab. 8). Also, from the results it is suggested that approach A has less usability.

#### *4.2.4 Task 3: Operating the integral system, regardless of its heterogeneity*

The subjects performed a simulated debugging routine, involving working simultaneously with both platforms. The assigned task was as follows: Data from the ConsoleIn RTC was read by the subjects (DF3). Then by using the keyboard, the read data was typed into the V-REP ROS node to test the movement of the simulated robot (DF4). Thereafter, the ConsoleIn RTC was closed and its source code was opened for editing and recompilation (VS2008), executing it again at the end (CP3) (see Fig. 10, note *III). The subjects proceeded as follows:

- Approach A: The subjects were required to find a way of visualizing the data from the RTC and to type it into the ROS node, as well as locating the RTC's source code. The subjects employed an additional RTC that displays the data from the ConsoleIn RTC, and used one of the V-REP ROS node's ROS services to input the data. Later, the subjects located the folder of the source code of the RTC, and proceeded to modify its variables through the VS2008. Then the ConsoleIn RTC was executed again.
- Approach B: The subjects used the HyperBot GUI to work with the system. The Role diagram in the System Developer Screen was used to visualize the data from the RTC by clicking on the element that represented its data port. Then by clicking on the element representing the required ROS service in the V-REP ROS node, the data was typed and sent. Later, by clicking on the element that represented the source code of the RTC, the GUI opened the file through the VS2008. After edition, the RTC was run again by clicking on the element that represented the executable file.

Compared with previous tasks, this one was of average difficulty. The same conclusion as in the previous task can be drawn (Tab. 9).

Table 9. Results for task 3: Operating the integral system, regardless of its heterogeneity. NE ratios are smaller than in the previous task (1.29 and 1.18), yet similar usability scores (27.0 and 78.3).

| Approach | SUS mean | SUS std. dev. | Usability | Learnability | Cronbach's alpha | Time mean [min.] | NE ratio |
|---|---|---|---|---|---|---|---|
| (A) Benchmark | 27.0 | 11.6 | 27.7 | 24.0 | 0.70 | 13.8 | 1.29 |
| (B) Proposed | 78.3 | 13.3 | 78.1 | 79.2 | 0.81 | 3.26 | 1.18 |

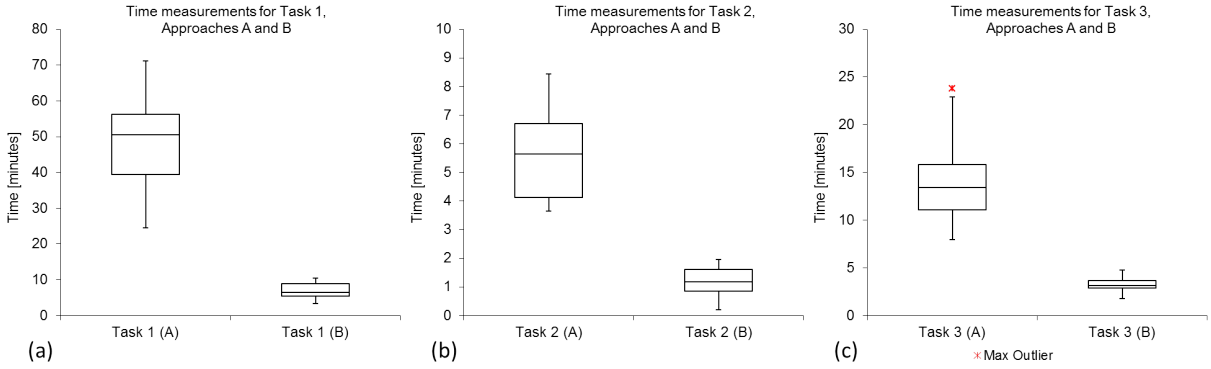


Figure 14. Comparison of tests' completion Time in tasks 1 (a), 2 (b) and 3 (c) for approaches A and B.

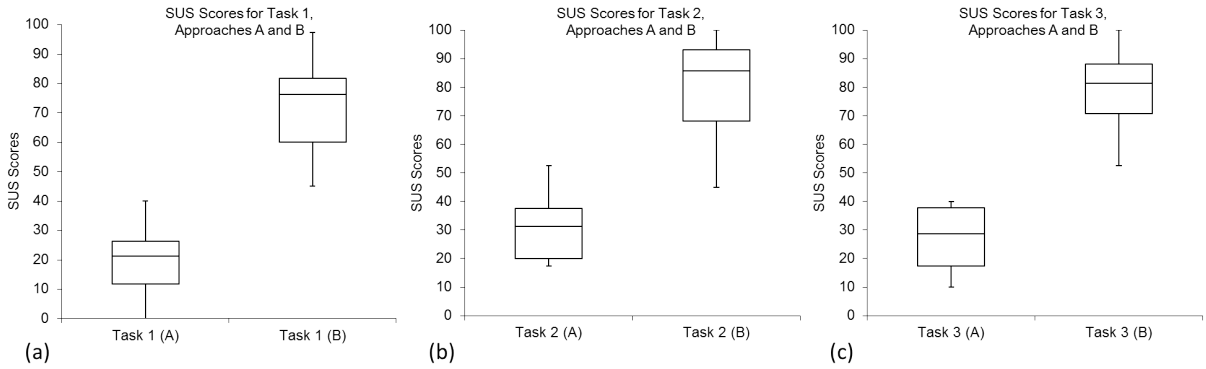


Figure 15. Comparison of tests' SUS scores in tasks 1 (a), 2 (b) and 3 (c) for approaches A and B.

### 4.3 *Results*

The procedure provided by our framework for normalizing abstractions at runtime was validated by a proof-of-concept through a case-study robotic system. The system behaved as expected, demonstrating that it is possible to compose the system by working with the proposed framework through the supporting infrastructure and the GUI. The result was a platform-agnostic model for the heterogeneous robotic system; its description was serialized and displayed into a diagram through the GUI, which was the front-end to operate the system.

In the usability validation, 12 subjects performed the normalization of abstractions at runtime and completed three basic administration tasks on a simplified version of the proof-of-concept system by using the benchmark approach (A) and our proposed approach (B); task completion time was measured and the subjects graded the tasks by using the System Usability Scale (SUS) (Fig. 14 and Fig. 15). The sample size of 12 subjects corresponds to having a margin of error of +/- 9.00 points around the reported SUS score, and to having an 80.0% chance of detecting a difference of 18.5 points or more (confidence level = 95.0%) between two average SUS Scores when compared using a within-subjects design test [40]. The SUS score samples had a Cronbach's alpha between 0.69 and 0.91, which can be considered to be inside an acceptable range [38]. By using analysis of variance (One-way ANOVA, alpha = 0.05), there was no statistical evidence

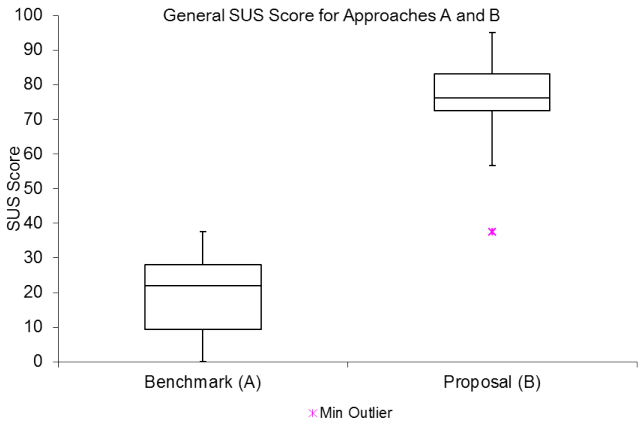


Figure 16. Comparison of General SUS scores between approach A and B.

Table 10. Results of the general evaluation. A comparison of both approaches' scores laid out a difference of 54.9 points in favor to our proposed approach (19.9 vs. 74.8). Also, it can be seen that our approach is both usable and learnable when the SUS score is decomposed in these two factors (75.4 and 72.4).

| Approach | SUS mean | SUS std. dev. | Usability | Learnability | Cronbach's alpha |
|---|---|---|---|---|---|
| (A) Benchmark | 19.9 | 11.6 | 20.7 | 16.7 | 0.76 |
| (B) Proposed | 74.8 | 14.8 | 75.4 | 72.4 | 0.84 |

that belonging to the expert or novice group will have an influence over SUS scores. Additionally, the subjects completed another SUS questionnaire, evaluating both approaches from a general perspective (Fig. 16, Tab. 10); the benchmark (A) scored 19.9 points and our proposal (B) 74.8 points. It is known that for this kind of usability test the average SUS score is 68.00 points, meaning that our proposal's score was higher than 72.58% of all usability tests overall. Thus, by using the percentile based Sauro and Lewis curve grading conversion [41], it translates into a B (lowest score is F and highest is A+). In conclusion, the results suggest that the usability of our proposed approach is superior when compared to the benchmark approach.

As for the time measurements, the proposed approach shows shorter task times (Fig. 14), especially for task 1, which involved the development of two relatively complex software programs for transferring data from one platform to another. The time taken for developing those programs (section 4.2.2, Fig. 14 (a)) was comparable to the time needed for composing the robotic system by using our approach (section 4.2.1, Fig. 13), though without the benefits that our approach provides. As complementary information, we roughly analyzed the Novice-Expert (NE) ratios to get an idea of the performance difference between novices and experts. However, to make a more in depth analysis on NE ratios, among other things, larger samples would be required.

### 4.4 *Discussion*

Our proposal makes it possible for the developers/integrators to choose features from a variety of platforms, while keeping the administration tasks simple and independent from the abstractions provided by the (strongly or weakly formalized) platforms, helping to converge on a unified solution. It also provides the means to make practical use of middleware platforms' similarities on-the-fly with relatively little effort, without needing to refactor (or even discard) the existing systems, or sacrificing in new systems the flexibility and features available natively inside specific platforms. Moreover, with our proposal the current programming methods could be enhanced when scripts and natural language processing algorithms are combined with our framework.

There are still improvements to be done to our proposed communication interface, for example, a current issue is the quite limited data transfer rate. While it is possible to stream relatively big

amounts of data, other approaches for bridging communications are still superior in this aspect. A possible solution could be the implementation of such existing approaches in our framework and communication infrastructure. Also, we found that although the subjects did not have a difficult time normalizing the abstractions by using the given templates, this process can benefit from a tool that automates this task. Such aspects may also impact usability.

In our opinion, the field of robotics middleware is just starting to mature, yet is still far from the ideal situation. Many core abstractions are converging, and some platforms are starting to provide interfaces that will make them compatible with other platforms. Some middleware platforms are being widely adopted in the robotics community (hobbyists, mainstream/academia developers). It happens that those platforms are conceptually similar, but due to their abstraction levels and special features, the way to administer them is different. An example of this is ROS and RTM. This reinforces our belief that variety in middleware will persist; it is necessary to provide the tools for intuitively sharing and using the knowledge contained in the developed software packages, like our proposal intends. Even if the moment of having a de-facto robotics middleware platform is reached, alternative platforms with different design policies and purposes must exist to keep evolving for the sake of improvement.

## 5. Conclusions

In this paper, we identified that by normalizing abstractions at runtime in highly-integrated heterogeneous robotic systems, the overall usability in the administration of their software and development tools can be improved, producing practical platform-agnostic representations of the robotic systems. We proposed the Framework for Integration of Elements and Resources by Roles (FIERRo), which allows the user to compose a system through the definition of "Roles", implemented in a runtime cross-platform infrastructure (ICPI) and its graphic interface (HyperBot). Our proposed framework complements the existing middleware platforms by providing a means for integrating the administration of features available in the various platforms.

The functionality of our approach was confirmed. Moreover, tests subjects performed trial tasks to evaluate the usability of our proposal. The results showed improved usability, and demonstrated the overall advantage of using the proposed framework.

Future work includes: application extensions of this framework through HyperBot (where the operation of additional screens will rely on FIERRo for the system abstractions), improvements on the communication infrastructure of the ICPI, as well as improvements on the Administration by Roles module. Further case studies in progress include the humanitarian demining robot GRYPHON [42]; it has a complex software system requiring constant changes for adapting it to new technology and ameliorations of the way tasks are done. Likewise, usability tests focused on this and other robots will be applied and analyzed.

## Acknowledgements

This work was partially supported by JSPS KAKENHI Grant Number 25303012. The first author acknowledges support from Instituto de Innovación y Transferencia de Tecnología ($I^2T^2$, N.L., MX), Consejo Nacional de Ciencia y Tecnología (CONACYT, MX) and the Roberto Rocca Education Program.

## References

[1] Benjamin M, Schmidt H, Newman P, Leonard J. Nested Autonomy for Unmanned Marine Vehicles with MOOS-IvP. J of Field Robotics 2010; 27(6):834–875

[2] Elkady A, Sobh T. Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. J of Robotics 2012; 2012:1–15
[3] Jeong Seok Kang, Young Gyu Kim, Hong Seong Park. Remote Data Transmission Middleware for Telerobotics. In: Proceedings of the International Conference on IT Convergence and Security; 2013. p. 1–4
[4] Kjærgaard M, Andersen N, Ravn O. DARC: Next generation decentralized control framework for robot applications. In: Proceedings of the IEEE International Conference on Control and Automation; 2013. p. 1598–1602
[5] Mizukawa M. Robot technology (RT) trend and standardization. In: Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts; 2005. p. 249–253
[6] Mohamed N, Al-Jaroodi J, Jawhar I. Middleware for Robotics: A Survey. In: Proceedings of the IEEE International Conference on Robotics, Automation, and Mechatronics; 2008. p. 736–742
[7] Mohamed N, Al-Jaroodi J. Characteristics of Middleware for Networked Collaborative Robots. International Symposium on Collaborative Technologies and Systems; 2008. p. 524–531
[8] Pigatto D, Freire Roberto G, Gonçalves L, Rodrigues Filho J, Roschildt Pinto A, Castelo Branco K. HAMSTER - Healthy, mobility and security-based data communication architecture for Unmanned Aircraft Systems. In: Proceedings of the International Conference on Unmanned Aircraft Systems; 2014. p. 52–63
[9] Planthaber S, Vogengesang J, Niessen E. CoHoN: A Middleware for Robots in Heterogeneous Communication Environments with Changing Topology. In: Proceedings of the IEEE International Conference on Robotics and Biomimetics; 2011. p. 2733–2738
[10] Gerkey B, Vaughan R, Howard A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the International Conference on Advanced Robotics; 2003. p. 317–323
[11] Tenorth M, Perzylo A, Lafrenz R, Beetz, M. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In: Proceedings of the IEEE International Conference on Robotics and Automation; 2012. p. 1284–1289
[12] Hunziker D, Mohanarajah G, Markus W, D'Andrea, R. Rapyuta: The RoboEarth Cloud Engine. In Proceedings of the IEEE International Conference on Robotics and Automation; 2013. p. 438–444
[13] Kato Y. et al. RSi-Cloud: Research and Development Environments for Robot Services using RSNP. The Institute of Electronics, Information and Communication Engineers Technical Report 2011; 111(178):61–66
[14] Matsuhira N, Namatame S, Ishida S, Yamaguchi T. Prototype development of robot system using RT middleware and RSNP network protocol. In: Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts; 2013. p. 45–50
[15] Smart W. Is a Common Middleware for Robotics Possible?. IROS 2007 workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware; 2007.
[16] Crick C, Jay G, Osentoski S, Jenkins O.C. ROS and Rosbridge: Roboticists out of the loop. In: Proceedings of the 7th ACM/IEEE International Conference on Human-Robot Interaction; 2012. p. 493–494
[17] Côté C, Brosseau Y, Létourneau D, Raïevsky C, François M. Robotic Software Integration Using MARIE. Intl J of Advanced Robotic Systems 2006; 3(1): 55–60
[18] Radestock M, Eisenbach S. Coordination in evolving systems. Trends in Distributed Systems: CORBA and Beyond; 1996. p. 162–176
[19] Mallet A, Pasteur C, Herrb M, Lemaignan S, Ingrand F. GenoM3: Building middleware-independent robotic components. In: Proceedings of the IEEE International Conference on Robotics and Automation; 2010. p. 4627–4632
[20] Bischoff R. et al. BRICS - Best practice in robotics. 41st International Symposium on Robotics; 2010. p. 968–975
[21] Basu A, Bozga M, Sifakis J. Modeling Heterogeneous Real-time Components in BIP. In: Proceedings of the IEEE International Conference on Software Engineering and Formal Methods; 2006. p. 3–12
[22] Bruyninckx H. The BRICS Component Model: A Model-Based Development Paradigm for Complex Robotics Software Systems. In: Proceedings of the 2013 Symposium On Applied Computing; 2013. p. 1758–1764
[23] Quigley M. et al. ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software; 2009.
[24] ROS [Internet]. Open Source Robotics Foundation. 2015 [cited 2015 May 20]. Available from:

http://www.ros.org/browse/list.php
[25] National Institute of Advanced Industrial Science and Technology, Japan [Internet]. OpenRTM-aist. 2013 [cited 2015 May 10]. Available from: http://www.openrtm.org
[26] Ando N, Suehiro T, Kitagaki K, Kotoku T, Woo-Keun Y. RT-middleware: distributed component middleware for RT (robot technology). In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems; 2005. p. 3933–3938
[27] Object Management Group [Internet]. Catalog of OMG CORBA/IIOP Specifications. 2012 [cited 2012 Jan 12]. Available from: http://www.omg.org/technology/documents/corba_spec_catalog.htm/
[28] Ceron Lopez A, Fukushima E, Kitano S, Endo G. Study of Framework Based on Roles for Application Development of Service Robots. In: Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts; 2013. p. 39–44
[29] Maurits L, Perfors A, Navarro D. Why are some word orders more common than others? A uniform information density account. In: Proceedings of 24th Annual Conference on Neural Information Processing Systems; 2010. p. 1–9
[30] W3C [Internet]. RDF Primer. 2004 [cited 2013 Feb 3]. Available from: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
[31] W3C [Internet]. SPARQL Query Language for RDF. 2013 [cited 2015 May 20]. Available from: http://www.w3.org/TR/rdf-sparql-query/
[32] Ceron Lopez A, Fukushima E. Proposal of Intelligent Cross-Platform Interface for Robotics Middleware. In: Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems; 2012. p. 382–387
[33] Ceron Lopez A, Fukushima E. Basic Study on Element Administration in Robotics Middleware 3rd Report: A GUI based on role definitions for using elements and resources in robotics software. In: Proceedings of The 32nd Annual Conference of the Robotics Society of Japan; 2014. p. 2A1-05
[34] Ueda K. et al. Development of HELIOS IX: An Arm-Equipped Tracked Vehicle. J of Robotics and Mechatronics 2011; 23(6):1031–1040
[35] Coppelia Robotics [Internet]. V-REP virtual robot experimentation platform. 2014 [cited 2014 Oct 15]. Available from: http://www.coppeliarobotics.com/resources.html
[36] Brooke J. SUS: A - quick and dirty - usability scale. Usability Evaluation in Industry; 1995. p. 189–194
[37] Lewis J, Sauro J. The Factor Structure of the System Usability Scale. In: Proceedings of the Human Computer Interaction International Conference; 2009. p. 94–103
[38] Tavakol M, Dennick R. Making sense of Cronbach's alpha. Intl J of Medical Education 2011; 2:53–55
[39] Urokohara H, Tanaka K, Furuta K, Honda M, Kurosu M. NEM: Novice Expert ratio Method - A Usability Evaluation Method to Generate a New Performance Measure. In: Proceedings CHI '00 Extended Abstracts on Human Factors in Computing Systems; 2000. p. 185–186
[40] Sauro J. A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices. Denver (CO): Measuring Usability LLC; 2011.
[41] Sauro J, Lewis J. Quantifying the user experence: Practical statistics for user research. Waltham (MA): Morgan Kaufmann; 2012.
[42] Fukushima E, Freese M, Matsuzawa T, Aibara T, Hirose S. Humanitarian Demining Robot Gryphon: Current Status and Objective Evaluation. Intl J on Smart Sensing and Intelligent Systems 2008; 1(3):735–753