# A Cross-Platform Communication Mechanism for ROS-Based Cyber-Physical System

Rui Zhao, Xu Tao, Davide Conzon, Enrico Ferrera
*LINKS Foundation*
via Pier Carlo Boggio 61,
Turin,
Italy
name.surname@linksfoundation.com

Yenchia Yu
*Tongji University*
4800 Cao'an Road,
Shanghai,
China
yuyenchia@tongji.edu.cn

***Abstract*—Recently, one of the main research topics in the context of application of Cyber-Physical System (CPS) in the Smart City and Industry 4.0 scenarios is the one related to the use of Robot Operating System (ROS)-based CPS. Specifically, one of the main interest is to allow a ROS-based smart robot communicating with other heterogeneous Internet of Things (IoT) applications in an intelligent environment to efficiently react to the system requirements and environment changes. However, the communication between the IoT systems will face many challenges and increase the cost and risks that lead to the requirement of a cross-platform communication for bridging the ROS-based CPS and other heterogeneous IoT applications.**

**This paper introduces ROS Edge Node for the interoperability between Robotics domain and other IoT domains, leveraging the highly modular BRAIN-IoT federation, which allows to decentralize, compose and dynamically federate the heterogeneous IoT platforms using OSGi specification, thanks to its dynamic modularity and wide usage in IoT middlewares. Together with the flexible integration with existing IoT devices/platforms within BRAIN-IoT platform, the event-driven asynchronous communication mechanism realizes cross-platform interaction with ROS-based CPS and solves the major challenges faced. This communication mechanism allows dynamic deployment of new functionalities for enhancing/extending the behaviour of robots according to external events. In addition, some specific behaviours to new "virgin" robots, which might be needed to extend the fleet of robots or replace damaged/low batteries ones can be dynamically deployed at the setup phase. In BRAIN-IoT platform, Edge Node behaves as IoT devices/platform adaptors which integrate the existing IoT devices/platforms. The ROS Edge Node is one type of the Edge Node, which bridges the underlying ROS-based robotics systems and BRAIN-IoT execution environment, thus communicates with various IoT systems connected to the BRAIN-IoT platform. A Service Robotic use case is developed to demonstrate the proposed solution, it shows how the ROS Edge Node enables the fast adaptivity and interoperability between heterogeneous IoT domains in a federated environment.**

***Index Terms*—Brain-IoT, Cyber-Physical System (CPS), Service Robotics, IoT Middleware, Cross-platform Communication, OSGi**

## I. Introduction

Nowadays, CPS are widely used in various aspects in our society, including but not limiting in areas such as manufacturing, energy, health, transportation and intelligent buildings, causing significant socio-economic impacts. CPS are defined as physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core. To realize such an intelligent system, various technologies need to be involved, including sensor and actuator technology etc. Together with IoT, Cloud computing and Cognitive computing, they become the main technologies of Industry 4.0 [1]. Specially, CPS and IoT are often discussed together, not only because they have many similarities but also because they are the foundations of the intelligent production environment, in another word, the smart factories, which is one of the most important topics of Industry 4.0.

The communication is an important research topic for both of CPS and IoT. As more and more different devices will be integrated in an intelligent production environment, the requirement of communication between different CPSs or between CPS and IoT devices are becoming more and more significant. For example, in a smart factory, different CPSs such as Autonomous Mobile Robots (AMR) [2] need to communicate with each other for cooperation, or an AMR needs to communicate with IoT devices such as automatic doors when it needs to cross different zones in the factory. However, such communication is not that easy to realize since heterogeneity is a basic property for both CPS and IoT devices.

On the one hand, heterogeneity exists because different CPS and IoT devices use different technologies in hardware, software, or communication method due to different needs. On the other hand, it is because the manufacturers of the devices are different. Different manufacturers will design the device according to their own standards. Devices from different manufacturers, even if they are using the same technologies, their protocol will be different. According to the latest statistics [3], there are officially 620 IoT platform companies in the global open market in 2019, in which 50% of the platforms focus on industrial use. From the perspective of communication, devices from these companies could use hundreds of different communication protocols which makes the standardization of communication protocol extremely difficult. In the recent years of research, the idea of using middleware to realize cross-platform communication is proposed.

This paper presents a novel adaptor, ROS Edge Node, which

implements an event-driven asynchronous cross-platform communication mechanism for ROS-based CPS. The work is a part of the H2020 research project BRAIN-IoT [4] that is a federation enabling the dynamic deployment, orchestration and monitoring of the distributed IoT applications leveraging the OSGi [5] technology, since OSGi is a series of specifications for Java dynamic modular system, it provides the dynamicity for the life cycle management of software components. One of the main functions is to make components as decoupled as possible, and to allow components to dynamically discover with each other, so that programmers can develop refinable, reusable, and assistive components in accordance with these specifications. Nowadays, OSGi is the widest used technology to support implementations of IoT abstraction layers. For example, Bosch ProSyst and Eurotech are two examples of companies providing OSGi-based IoT gateways; Oracle® Fusion Middleware [6] is developed for developing Java Enterprise Edition (EE) management applications for Oracle WebLogic Server; SNPS [7] is an OSGi-based middleware for Wireless Sensor Networks; The OSGi-based software platform Eclipse SensiNact [8] provides support in technical aspects (e.g. Connectivity, Interoperability, Data processing, and Developer Tool) related to smart city platforms. BRAIN-IoT platform is implemented with OSGi and it aims integrating with generic existing IoT devices or IoT platforms or existing IoT middlewares to allow them communicating with each other. ROS Edge Node is one implementation of the BRAIN-IoT Edge Node mainly focused on the integration of robotics platforms in IoT domain allowing to interoperate with other heterogeneous IoT devices by mapping the ROS services to OSGi services, to maximize the connectivity of robots within IoT systems. ROS Edge Node is a modular software component of the BRAIN-Io platform. It provides four main features: 1) *Interoperability*: it provides the connectivity to IoT platforms connected to BRAIN-IoT solution. 2) *Plug & Play*: leveraging the OSGi specification, the component follows an event-driven approach and it is developed as a software module that can be deployed/undeployed at runtime without interrupting other running services. 3) Automatic Adaptation, it provides a code generator to automatically expose the ad-hoc ROS services provided by different ROS-based CPSs and speeds up the adaptor development process. 4) Standards Compliant, it exploits the Web of Things (WoT) Thing Description (TD) [9] describing the services provided by the ROS-based CPS, making it more portable to the production environment, not restrict to the OSGi implementation.

The rest of this paper is organized as follows. Section II introduces the state of the art of the solutions similar to the one proposed in this paper in multiple domains and the challenges to be addressed typically by these software. It also outlines the relevant technologies used to develop the proposed solution. Section III describes the architecture of the ROS Edge Node, functionalities and development process. Section IV implements a simple robotic application to validate the solution in a distributed environment. Finally, the paper concludes with the Section V to provide a summary of the cross-platform communication mechanism for ROS-based CPS and the future work being undertaken.

## II. BACKGROUND

### A. State of the Art

One of the most popular example of CPSs is represented by smart industrial robots. This sector is really fragmented, in terms of hardware and software architectures, But in recent years, more and more robots have begun adopting one common technology, the middleware ROS. According to the annual ROS Metrics Report for 2019 [10], there are near 150 types of documented robots available to the community with ROS drivers, and in recent years, the total number of papers citing "ROS: an opensource Robot Operating System" (Quigley et al.,2009) increase at an annual rate of 20% to 30%. More and more famous robot vendors such as Asea Brown Boveri (ABB) Ltd., Comau Spa. and Kuka AG are starting to support ROS in some models of their robots. ROS is the widest used abstraction layer for robotics. However, even the robots are using ROS middleware, the problems in communication still exists. ROS-based CPS are often used to communicate with other heterogeneous IoT applications to satisfy system requirements and react physical environment changes. There are two existing middlewares for interacting with ROS-based CPS.

**The ROS–YARP Framework for Middleware Interoperability** [11]: ROS and Yet Another Robot Platform (YARP) [12] are the most popular robotics middlewares. YARP is more used in the domain of humanoid robots and developmental robotics, whereas ROS has higher focus on mobile robots. They have complementary functions and many robotic platforms may benefit from using functions from both. But it's not an easy task mainly due to fundamental differences in the communication architecture. This approach generates the "bridging gap" code from a configuration file, connecting YARP ports and ROS topics through code-generated YARP bottles. It supports YARP/ROS and viceversa sender/receiver configurations. Reading from/sending to ROS topics needs an additional conversion, which is handled by the existing runtime YARP to ROS converter. The generator abstracts YARP and ROS developers from dealing directly with interoperability issues. However, this work is only focusing the interoperability among robotics middlewares.

**ROBIN Middleware for CPS** [13]: ROBIN is a middleware funded by European H2020 program providing an effective, bidirectional, reliable and structured data interchange mechanism to address the demand for flexible robotics in contemporary industrial environments and the necessity to integrate robots and automation equipment in an efficient manner. ROBIN, the robotics bridge to industrial automation, aims to allow the interoperability between robotics and automation systems by enabling the communication between ROS and CODESYS [14], which is a softPLC[1], a real-time multi-task control kernel, that can run on embedded devices

[1]http://www.softplc.com/

and that supports a variety of fieldbuses, and other industrial network protocols. CODESYS handles the establishment of external communication with the equipment via a fieldbus and communicates with ROS through the developed robotics bridge. More specifically, since ROS implementation follows the publisher–subscriber messaging pattern, which enables exchanging data between ROS nodes [15], A ROS node is basically a process that performs computation and it is an executable program running inside the robotics application. Many nodes can be implemented in ROS packages. The proposed bridging mechanism allows developers to create or include in existing ROS packages the valuable feature of connecting with an external device via fieldbuses or industrial network protocols promoted by the softPLC bridge. Multiple ROS nodes can access and modify the data shared with the softPLC application in ROS. The information to be propagated to the external devices could be published on a ROS topic, handled by the developed bridge, and then relayed by CODESYS to the proper industrial network protocol or fieldbus.

### *B. Problems with State of the Art*

ROS allows the communication between heterogeneous devices, being deployable on heterogeneous platforms. After ROS is deployed on the device, it can use ROS as communication method to communicate. However, it can only support communication between devices developed based on ROS, it cannot be used to communicate with off-the-shell devices using different technologies. Furthermore, the messages used in ROS are the original data sent by the device, which has not been standardized. The receiver must know the content of the communication in advance, otherwise it will not be able to understand the received data. In State Of The Art (SOTA), one of the limitations of ROS–YARP middleware is the applicability area, which is limited to ROS and YARP only, while CPSs are widely used in various aspects. The robotic bridge provided by ROBIN project allows the communication between ROS and the automation application through the inter-node communication mechanism in ROS, but it requires the developers to directly create or include in existing ROS packages the valuable features for interacting with other external devices via fieldbuses or industrial network protocols. This requires to the developers to be expert in ROS programming. Besides, the direct operation on existing ROS packages may bring the risk of the damaging the basic functionalities. Moreover, in the real production environment, the robotics applications are significantly sophisticated and dynamic, requiring to the bridge to be flexible enough to react to the continuously changing production environment; to allow this the robotic bridge needs to support the update at runtime and the deployment on demand and this feature is not feasible using only native ROS.

Instead, ROS Edge Node bridges ROS and other IoT platforms using OSGi specification, in such way, the robot's behaviours can be developed at the application level using OSGi instead of ROS. And this allows to have all the advanced OSGi capabilities applied in robotics scenario(i.e., start and stop atomic behaviours at runtime, deploy new ones, import, update and upgrade behaviours at runtime every time a new more stable or more secure version is available).

### *C. Challenges*

ROS Edge Node intends to address the gaps mentioned in the previous section in the current State of The Art (SoTA) and the typical challenges faced in researches on middleware for CPS [16], [17]. More specifically, apart from supporting the interoperability between the ROS-based CPS and other heterogeneous IoT applications, it also supports adaptivity, security and privacy protection and autonomous operation, as explained in the following subsections.

**Abstraction and Automatic Adaptation**: [18] An ideal middleware for an intelligent environment such as the IoT should provide abstractions at various levels such as heterogeneous input and output hardware devices, hardware and software interfaces, data streams, physicality and the development process. And an Adaptive middleware is usually motivated by the need of adapting the middleware to changes in application's requirements, changes of environmental conditions, fixing middleware's bugs or extending/improving the middleware functionality. ROS Edge Node solution proposes an approach to create a software component to abstract the ROS-based CPS for communicating with other OSGi-based IoT middlewares/applications through the distributed BRAIN-IoT EventBus. The adaptor can be generated according to different ROS platform implementations. For the simplicity, a code generator is provided to speed up the development process.

**Security and Privacy**: Automatic communication of real-life objects represents a huge challenge in terms of trust, security and privacy. The security and privacy component is needed to provide the integrity of the collected data (stream) and to ensure that the user's privacy is not violated. The data can only be able to connect to authenticated/certified IoT devices. The management support of security and privacy has to be considered as a main function of the middleware for the IoT. The ROS Edge Node adaptor will be integrated with the BRAIN-IoT platform, which introduces a holistic end-to-end trust framework and privacy-awareness and control approach to address the challenge [4]. Currently, the integration between ROS Edge Node and the end-to-end framework guarantees only authenticated ROS-based CPSs can integrate with BRAIN-IoT platform and communicate with other authenticated IoT systems.

**Autonomous Operation**: Many CPS applications are considered complex systems which can be in a huge number of different states at any point of time. It is generally extremely difficult to develop code to handle all these states effectively and in a timely manner. Having middleware that supports autonomous operations such as self-adaptive, self-resilient, and self-protected services can relax implementing and operating these complex CPS applications. The ROS Edge Node addresses this challenge supporting the development of complex IoT solutions for monitoring and controlling physical environ-

ments and systems. Specifically, it simplifies the operation of application providing a an event-driven notification method, which allow avoiding to use polling methods to query the robot or mission status, reducing greatly the network traffics.

### D. Relevant Technologies

**RosJava Open Source Library**[2]: RosJava project provides a pure Java implementation of ROS, and it also can interconnect to an existing ROS environment through the Internet Protocol (IP) address. It provides a client library for ROS communications in java that allows Java programmers to quickly interface with ROS topics, services and parameters through the eXtensible Markup Language (XML)-Remote Procedure Call (RPC) [19] protocol. It provides some common Java API allowing to create new ROS nodes, services, topics in native ROS environment, and the corresponding ROS clients. The library can be fully integrated in OSGi software.

**JCodeModel Open Source Library**[3]: JCodeModel is a Java code generation library. It provides common API to generate Java programs using Java language.

**World Wide Web Consortium (W3C) and WoT**: In recent years, W3C organization has developed the WoT [9] standard aiming to achieve interoperability problem between IoT platforms and application domains. WoT provides a mechanism for describing IoT interfaces, allowing IoT devices (physical or virtual entity) and services to communicate with each other, independent of their underlying implementation, and can span multiple network protocols. In addition, WoT also provides a standardized way to define and plan IoT behaviors. WoT Architecture specification is centered on the scope of W3C WoT standardization, divided into several building blocks. The four core building blocks provided by W3C WoT are: *Thing Description*, *Binding Template*, *Scripting Application Programming Interface (API)*, *Security and Privacy Guidelines*. More specifically, The central building block is the WoT TD[4], which can describe the metadata of the object and the network-oriented interfaces and it's the entry point of a Thing. TDs are encoded in a JavaScript Object Notation (JSON) format that also allows JSON-based Serialization for Linked Data (JSON-LD) processing, primarily intended to be a way to use Linked Data in Web-based programming environments. The building blocks allow an application client (a Consumer) to interact with Things that expose diverse protocols through the three types of *Interaction Affordances* defined by W3C WoT Interaction Model representing the capabilities of individual Things: i) *Properties (PropertyAffordance class)* can be used for sensing and controlling parameters, such as getting the current value or setting an operation state; ii) *Actions (ActionAffordance class)* model invocation of physical (and hence time-consuming) processes, but can also be used to abstract RPC-like calls of existing platforms. iii) *Events (EventAffordance class)* are used for the push model of communication where notifications, discrete events, or streams of values are sent asynchronously to the receiver. TD can be used for flexible implementation and simulation (if required). WoT will break the barrier of interoperability of various IoT platforms, thereby contributing to the explosive growth of the market. It doesn't aim to define a new platform, but to use the metadata to bridge existing platforms and standards.

In this paper, these technologies will be used in the following aspects. The ROS Edge Node, will be developed as OSGi bundles, which can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot in BRAIN-IoT federation.

RosJava can be considered as the bridge between ROS world and Java world. It provides an efficient way for the ROS Edge Node to establish a communication with ROS-based devices. Different ROS functionalities will be mapped into different OSGi services in the ROS Edge Node. The mapping procedure will be done automatically through JCodeModel library with a TD of the underlying ROS environment. Anyway, the corresponding formatting procedure of events and integration with BRAIN-IoT framework should be done by developers.

## III. Architecture

This section details the ROS Edge Node architecture along with its role in the overall BRAIN-IoT platform.

### A. Overview in BRAIN-IoT Context

ROS Edge Node will be integrated with BRAIN-IoT Fabric [20] infrastructure service, which is composed with a set of the computing resources (physical/virtual machines) and provides a distributed OSGi execution environment allowing the interaction between the OSGi services deployed on it through events, thanks to the implementation of OSGi Alliance specifications for Remote Services and Remote Service Admin [5]. Each BRAIN-IoT Fibre is an OSGi R7 framework, and different Brain-IoT OSGi bundles deployed on local and remote Fibres can communicate with each other using some specific strongly typed BRAIN-IoT events delivered in the asynchronous BRAIN-IoT EventBus [20] according to the BRAIN-IoT approach. The BRAIN-IoT Nodes are the Service Fabric Fibres with different BRAIN-IoT Services deployed on them, they can be of two types: BRAIN-IoT Processing Nodes and BRAIN-IoT Edge Nodes. The first ones are a sort of Service Fabric Fibres deploying IoT application logic and controlling the CPS behaviours through their adaptors supported by the machine learning algorithms. The latter ones are Fabric Fibres with the installed edge components deployed on the top of them. BRAIN-IoT Fabric allows users to label the BRAIN-IoT nodes, thus to guide where the BRAIN-IoT service, satifying the required capabilities, should be deployed at runtime. The BRAIN-IoT architecture allows the dynamic redeployment of 1) new functionalities for enhancing/extending the behaviour of robots according to external events. 2) specific behaviours to new "virgin" robots, which might be needed to extend the fleet of robots or replace demaged/low batteries ones. Each ROS Edge node can be considered as an access point or an adaptor to ROS-based CPS, to allow

[2]https://github.com/rosjava

[3]https://github.com/phax/jcodemodel

[4]https://w3c.github.io/wot-thing-description/

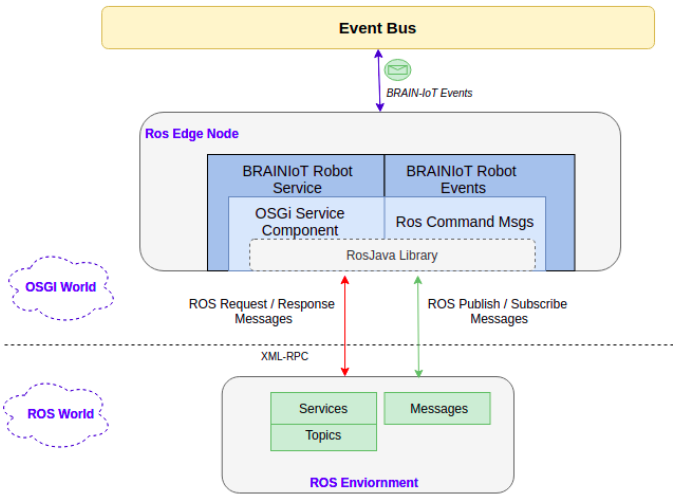


Fig. 1. ROS Edge Node structure

heterogeneous IoT applications running on the processing nodes to control the robots. In the BRAIN-IoT platform, the authors' contribution is to develop the ROS Edge Node as OSGi Declarative Service, so that from the northbound it can receive the interested BRAIN-IoT events from other different IoT platforms in a distributed environment, then construct the data and send to the connected ROS environment. Therefore, any new functionalities for enhancing/extending the behaviour of robots according to external events can be developed using OSGi instead of ROS. From the southbound, the ROS Edge Node is able to retrieve the information from ROS and inject to the BRAIN-IoT Fabric as BRAIN-IoT events, which will be received by other BRAIN-IoT services. The ROS Edge Node aims to achieve the interoperability between heterogeneous IoT applications integrated within BRAIN-IoT platform and the ROS-based CPSs. It exposes the ROS functionalities as OSGi services.

The architecture of ROS Edge Node is shown in Fig.1. For its development and validation, the authors have used the ROS simulation for BRAIN-IoT service robotic use case provided by the Robotnik Automation S.L.L. (see Section IV). The ROS Edge Node has two main requirements: i) to expose all the relevant ROS functionalities as OSGi services (task done by *OSGi Service Component*). The objective is to make the adaptor able to interoperate with the ROS environment leveraging APIs provided by the open source Rosjava project, in this way, the adaptor is able to send/receive the ROS request/response messages from/to the ROS services and publish/subscribe to the ROS topics between the OSGi world and ROS world. ii) To collect and format of the BRAIN-IoT events from different sources based on the Publish-Subscribe pattern[5] (task done by *BRAIN-IoT Robot Service*). The adaptor receives the events from heterogeneous platforms in the distributed BRAIN-IoT Fabric environment and constructs them as Java objects representing ROS messages, then transforms to ROS environment through the exposed OSGi services. In contrast, the adaptor also retrieves the ROS messages from the native services/topics and convert to the BRAIN-IoT events, then deliver them in the distributed EventBus. The connectivity with ROS is configurable through the ROS environment variable *ROS_MASTER_URI* by default whose value is configurable on-the-fly in order to set up which machine as a ROS master when a new "virgin" robot join the fleet of robots.

[5]https://en.wikipedia.org/wiki/Publish\%E2\%80\%93subscribe_pattern

### B. ROS Edge Node Approach

The ROS Edge Node adaptor is further explained in this subsection, detailing its approach, implementation steps and the addressed challenges.

*1) Abstraction of ROS environment:* It aims to address the interoperability challenge when integrating other IoT devices in BRAIN-IoT platofrm. As Fig.1 shows, the basic function of ROS Edge Node is to map the ROS messages to BRAIN-IoT events and vice-versa. The BRAIN-IoT services interact with each other by leveraging the Requirements and Capabilities metadata provided by default by all OSGi Bundles, the events issued by a *source* BRAIN-IoT service contains the sufficient information presenting the Requirements of this service, in the meanwhile, the events also identify the Capabilities of the *sink* BRAIN-IoT services that will consume them.

The ROS Edge Node simply interconnects to the ROS environment thanks to the RosJava library which provides a simple interface enabling the ROS Edge Node to automatically generate a set of Java object classes, which is totally compliant with native ROS messages' structure in the ROS environment. The instances of the classes could be transformed to the ROS environment. However, it's not possible to make the BRAIN-IoT events types the same as the names of the native ROS messages. For a single robot, there could be hundreds of types of native ROS messages in the ROS environment and their data structures are in general very complex, direct mapping between the ROS messages types and BRAIN-IoT events is inefficient and will increase the complexity of the usage of the events for other IoT applications. Also, different robots having same functions may use completely different types of ROS messages as commands. In such case, if directly mapping each ROS message into an event, the difficulty to integrate different robots in a system will be increased. Thus, some common data types including necessary information need to be defined and shared between diverse IoT applications. So it's necessary for ROS Edge Node to format the received events into the specific Java objects that are compliant with ROS messages.

The ROS Edge Node wraps also the native ROS functionalities, exposing them to the OSGi services for heterogeneous IoT applications' access, by creating the corresponding ROS services and publish/subscribe clients in OSGi bundles, through the APIs provided by RosJava project. The exposed OSGi services are a set of java classes containing multiple robot operations mapped to Java methods. The *BRAIN-IoT Robot Service* in Fig.1 is a wrapper of the exposed OSGi services, with the specific Capabilities information represented by the

consumed event types. The ROS Edge Node is responsible for communicating with other BRAIN-IoT services using Events. When the robot service receives an event from the EventBus, it will construct a ROS message in Java type and perform the corresponding action by calling the exposed ROS functionalities. Thanks to the BRAIN-IoT solution, the service will be deployed on the BRAIN-IoT Fabric by the event-driven mechanism. It's completely de-coupled from the underlying BRAIN-IoT Fabric runtime.

*2) The Autonomous Operation:* As mentioned in Section II-C, supporting autonomous operation is one of the challenges for a CPS adaptor. In the ROS Edge Node, the autonomous operation is fully supported by a feedback mechanism: the ROS Edge Node is responsible for continuously querying the execution status of CPS where it is installed and then to issue an response event if the status changes. This allows building services, which leverage the ROS Edge Node, which are "smarter" and reducing the communication workload on the EventBus,(see Fig.6 in Section IV).

*3) Automatic Adaptation and Standards Compliant:* This is a usual task that needs to be supported by such type of solutions. For ROS-based devices, this challenge is addressed by ROS Edge Node. As mentioned above, the exposed service components are a set of java classes containing multiple methods for robot operations. The operations are done through ROS services/topics clients in Java code via simple API provided by RosJava library. Normally, when developers create the java clients for the used native ROS functionalities, the java clients could be grouped in one or more Java classes for better organization.

Since the services or topics and their related methods in a component class have the same structures, the authors achieve to automatically realize the adaption by creating a Code Generator to automatically generate the OSGi service classes from a predefined configuration file as the input. Since all entities in ROS-based CPS communicate through services and topics, the authors choose to use the W3C WoT TD, which is a general standard for both integrating diverse devices and interoperability of diverse applications to describe the ROS functionalities. In the proposed solution, WoT TD describes the interfaces for OSGi to expose the ROS services and topics. In this way, a standard approach is used to describe the ROS API and to generate the corresponding OSGi services, this allows to have a solution WoT integration-ready and highly reusable. The ROS functionalities are compliant with TD specification: 1) the topics are as *Properties Interaction Affordances* 2) the services are described as *Actions Interaction Affordances*.

For a ROS service, the part of TD file is shown as Fig.2. More specially, there could be a set of ROS services described in the *Actions* element, and for each of them, the information will include *serviceClientName*, *serviceName*, *serviceType*, *serviceRequestType*, *serviceResponseType* and *ClassName*, which are used by the Code Generator to create service clients in OSGi world. The Code Generator uses the *ClassName* property to generate multiple component classes

```
{
   "@type":"Thing",
   "properties":{...},
   "actions":{
      "serviceClientName":{
         ...
            "serviceName":{...},
            "serviceType":{...},
            "serviceRequestType":{...},
            "serviceResponseType":{...},
            "ClassName":{...}
         },
         ...
   }
}
```

Fig. 2. Thing Description of ROS Environment

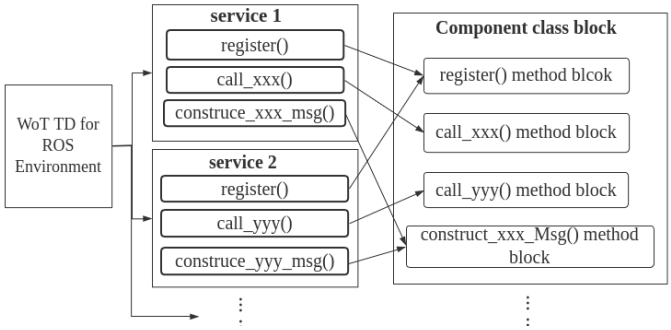


Fig. 3. Expose of ROS Environment to OSGI Services Using WoT TD

for different types of robot functionalities, and each component can contain multiple service clients and the different operations could be done through the methods called by each client. The values of other properties will be used as the arguments of the generated methods. Similarly, the ROS topics will be described in the *Properties* element, including the information about *Role*, *ReferenceName*, *TopicName*, *TopicType*, *MessageType* and *ClassName*. The value of the *Role* property could be *publisher* or *subscriber*, so the corresponding client type will be created with the client name equal to the value of *ReferenceName*.

To automatically expose relevant ROS functionalities and speed up the ROS Edge Node development for the ad-hoc ROS platforms, A code generator is provided to generate the artefacts automatically by taking the TD as a configuration file, to describe the ROS services and topics, as shown in Fig.3, which is demonstrated in Fig.5 in Section IV. This approach greatly reduces the dependence for developers in the process of adapting to different ROS-based CPSs, increasing the development simplicity. Developers can therefore focus more on the development of BRAIN-IoT services rather than on adaptation work.

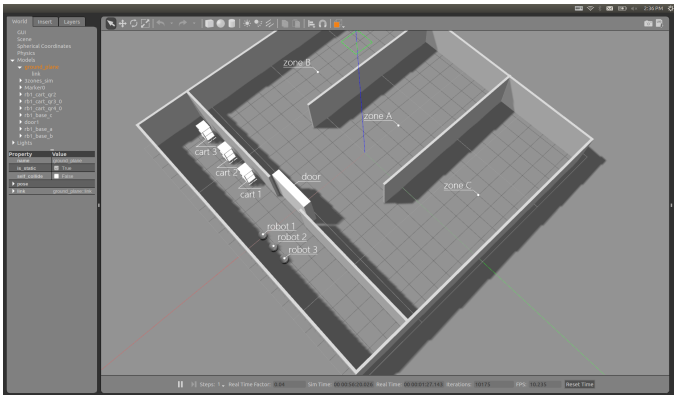


Fig. 4. Warehouse simulation

| Component example based on ROS services | | |
|---|---|---|
| Component | Include services | Related methods |
| GoToComponent | gotoRun | register() |
| | | call_ gotoRun () |
| | | construct_ gotoRun_Msg() |
| | gotoCancel | register() |
| | | call_ gotoCancel() |
| | | construct_ gotoCancel _Msg() |
| | gotoQuery | register() |
| | | call_gotoQuery() |
| | | construct_ gotoQuery _Msg() |
| Component example based on ROS topics | | |
| Component | Include topics | Related methods |
| AvailibilityComponent | availability (subscriber) | register() |
| | | get_value() |

Fig. 5. Method Blocks of Automatically Exposed ROS Services from TD file

## IV. Use Case Demonstration

In this section, a brief description of ROS simulation used for the Brain-IoT Service Robotic Use Case and the result obtained evaluating the cross-platform communication mechanism using it are presented.

### A. Use Case Introduction

Smart warehouse is a popular application of Industry 4.0. The basic elements of a smart warehouse include AMR, heterogeneous IoT devices and cargo carts. To be "smart", these elements need to interact and cooperate with each other. The use case aims to realize one of the basic function of smart warehouse: the cart movement. The Fig.4 shows the warehouse with three zones in 3D perspective in Gazebo simulator: a docking area, a unloading area and a storage area. The last two zones are separated by an automatic door, which is an IoT device. There are three *rb1* base robots (robot1, robot2 and robot3) in the docking area responsible for moving all carts to the storage area. In the unloading area there are 3 carts (cart1, cart2 and cart3) and each has a different QR code attached. The storage area is divided into three sub-zones (zone A, zone B and zone C). The robots need to pick up these carts from the unloading area, pass through the door and place them in the storage area according to the received commands from the robot application.

### B. Robot System Design

In order to demonstrate how the adaptor bridges the ROS environment and the OSGi environment, the authors implemented a simple multi-agent robot system based on OSGi to control the simulated robots.

The system contains three system parts: a Robot Behaviour Service, a ROS Edge Node Service and a Door Edge Node Service. There are several tables storing the coordinates of the carts and the storage positions will be used as the shared resources among three robots. In the ROS simulation, the basic task for each robot is that starting from the docking area, it needs to go to the picking area for picking up a cart, then pass through a door to the storage area and finally place the cart, the procedure is controlled by the Robot Behaviour Service. The new functionalities for enhancing/extending the behaviour of robots according to external events can be dynamically upgraded and redeployed in the BRAIN-IT architecture. Based on that, the authors define the events in three types: action, query and cancellation. The events in action type includes *WriteGoTo*, *PickCart* and *PlaceCart*, which present the basic functions of the robot. There are also other events defined for querying the execution status of corresponding actions and for cancelling current actions.

In the use case, there are three services related to the *WriteGoTo* action event, a WoT TD file is defined for the communication with the robotic system, the code generator using the TD as input will automatically generate the class named *GoToComponent* as shown in Fig.5 by using the open source JCodeModel library which is a Java code generation library. Specifically, according to the ROS functionalities described in the TD, there are three ROS service clients (e.g. gotoRun, gotoCancel, gotoQuery) will be created in the generated class. when the ROS Edge Node receives an event, the corresponding *construct_XXX_Msg* method representing the operation of the client will be called to construct a Java object representing the ROS message as a ROS service request to be sent to the native ROS environment through the *call_XXX* method of the service client, where the *XXX* stands for the name of the service client.

The Robot Behaviour Service continuously checks the shared tables to get a task and then it controls the robots through a sequence of BRAIN-IoT events to finish the mission. The events will be received by the ROS Edge Node Service and sent to the connected robot. As an example shown in Fig.6, after the ROS Edge Node receives a *WriteGoTo* event from the Controller Service containing the coordinates where the robot should go, with the autonomous operation of the ROS Edge Node, only twice communications are needed between the Controller Service and the ROS Edge Node, one for sending the action command, while the other one for receiving the action result. When the robots detected a door on the way to the storage area, it reports the situation and the Robot Behaviour Service will instruct the Door Edge Node Service to open the door.

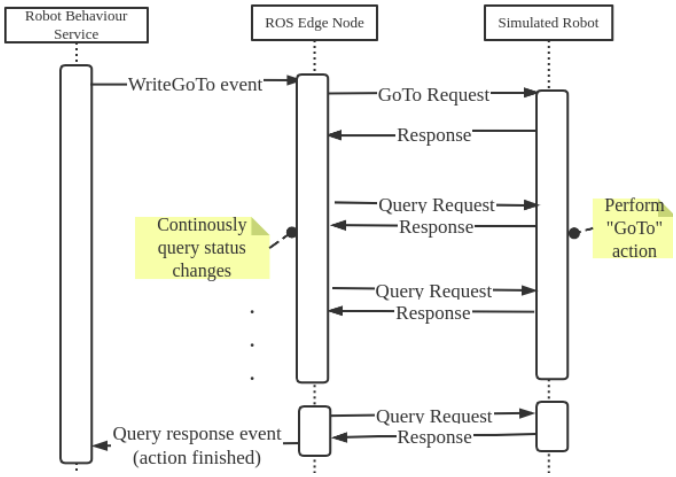


Fig. 6. Communication Procedure of "WriteGoTo" action for ROS Edge Node

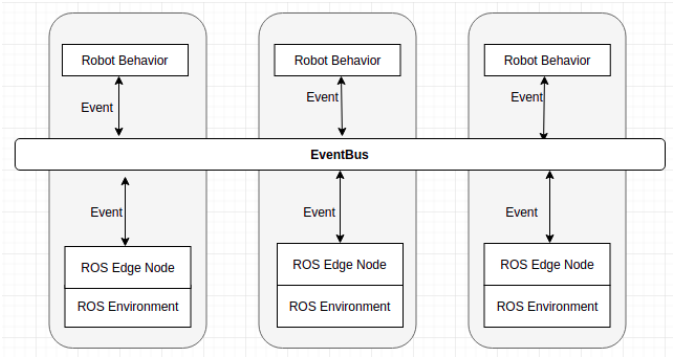


Fig. 7. Deployment of Robot System in the distributed BRAIN-IoT Fabric

In the test, the robot system above is deployed on a distributed Fabric environment containing multiple Raspberry Pis and one Linux server running the ROS simulation as BRAIN-IoT nodes in a same network. The Linux server is labeled in advance when BRAIN-IoT Fabric cluster is created and the ROS Edge Node Service is automatically deployed on it when the label is detected. Since there are three simulated robots, three instances of the ROS Edge Node Service instantiated by OSGi Configuration Admin Service Specification [5] to connect to the robots through IP address, the interested events will be delivered to the ROS Edge Node through the integrated EventBus. In the multi-agent system, each Robot Behaviour Service instance will control one robot to take a task from the shared table, after one task is finished, it will start next iteration. In this case, the deployment of Robot Behaviour Service and the ROS Edge Node Service in the real physical environment can be shown as Fig.7. Meanwhile, The door in the simulation environment is a IoT device controlled by the Door Edge Node Service, which is simulated in the ROS environment.

### C. Validation in Simulation Environment

When the multi-agent robot system is activated, each Robot Behavior Service instance will check the shared cart table to find the pending tasks. When a cart to be moved is found in the table, Robot Behavior will change the status of the cart to *moving* and start the moving procedure. Robot behaviors will deliver a sequence of events to the corresponding ROS Edge Node one by one to command the robot to move to cart position, pick up the cart, move the cart to specific position in the storage area and finally drop the cart. When ROS Edge Node receives an event, it extracts the information contained in the event, constructs a ROS message in Java type, and communicates with the robot through the exposed OSGi services. ROS Edge Node is responsible for continuously querying the execution status from the specific ROS service/topic, when the action of an event is finished or something wrong happened, a feedback event will be issued to inform the Robot Behavior to ask for the next action. During the moving procedure, if the door is closed, it will scan the QR code of the automated door, and return a *DoorFound* event to the Robot Behavior. The Robot Behaviour will send a *OpenDoor* event to the Door Edge Node service to open the door.

After finishing the moving procedure, the Robot Behavior Service will change the status of the cart to *moved* in the table and search for the next mission. Finally all carts are moved in ROS simulation.

### D. Validation in Physical Environment

To prove the feasibility of the approach proposed in the paper, ROS Edge node has been installed on a real *rb-1 base* mobile robot. Due to the difference of the simulated world and the physical world, the autonomous robotic system is not used in the physical environment, but authors implemented an Orchestrator service, which provides a simple interface for users to manually send a sequence of BRAIN-IoT events including the coordinates to control the actions of the robots in a physical environment. The Orchestrator is implemented as a OSGi Declarative Service and it injects a sort of commands in the Apache Felix Gogo [6], which is a subproject of Apache Felix implementing a command line shell for OSGi, which could be accessed via its web interface in a distributed BRAIN-IoT Fabric environment. When a user enters the command in Gogo shell, the corresponding method in the Orchestrator will issue a specific event to EventBus. The Orchestrator service could be installed on any BRAIN-IoT node. Finally, ROS Edge Node is able to receive the events and the robot will move towards to the target positions.

## V. Conclusion and Future Work

The paper has presented the ROS Edge Node, which enables the interoperability between the ROS-based CPS applications and other heterogeneous IoT platforms in a sophisticated IoT software ecosystem, based on the available services in BRAIN-IoT framework. This solution provides several innovative features, to ease such interaction. Firstly, it can be dynamically deployed and flexibly scaled on demand to connect to multiple ROS-based CPSs at runtime whenever a new CPS joins the

[6]https://felix.apache.org/documentation/subprojects/apache-felix-gogo.html

cluster. Secondly, it provides an approach to automatically expose the ROS functionalities as OSGi services for bridging the ROS world and OSGi world. Thirdly, ROS Edge Node maximizes the flexibility of the mechanism by supporting customized autonomous operation to detect the changes of action status and issue feedback events, thus greatly reduces the communication load on EventBus and its dependence on other system components. Finally, the use of WoT TD standardizes the description of the ROS functionalities.

Compared with the existing CPS middleware or IoT middleware, the proposed solution presents several advantages, but some further developments are still ongoing. The integration with the BRAIN-IoT End-to-End Security Framework are still under development. Besides, its functionalities will be enriched more, such as the graphical monitoring of robot status and the warehouse coordinates at runtime by integrating with other BRAIN-IoT monitoring tools. ROS Edge Node will be tested on the real robots with a more complex Robotics use case and the performance will be compared with other existing solutions, in the future work.

## ACKNOWLEDGMENT

The work presented here was part of the project "Brain-IoT-model-Based fRamework for dependable sensing and Actuation in iNtelligent decentralized IoT systems" and received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780089.

The simulation environment is provided by Robotnik Automation S.L.L.

## REFERENCES

[1] N. Boulila, "Cyber-physical systems and industry 4.0: Properties, structure, communication, and behavior," 04 2019.

[2] S. Barai, M. K. Kundu, and B. Sau, "Path following of autonomous mobile robot with distance measurement using rfid tags," in *2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR)*, 2019, pp. A3–4–1–A3–4–4.

[3] nud Lasse Lueth, "Iot platform companies landscape 2019/2020: 620 iot platforms globally," https://iot-analytics.com/iot-platform-companies-landscape-2020/, December 2019.

[4] D. Conzon, M. R. A. Rashid, X. Tao, A. Soriano, R. Nicholson, and E. Ferrera, "Brain-iot: Model-based framework for dependable sensing and actuation in intelligent decentralized iot systems," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–8.

[5] O. Alliance, "Osgi core release 7," OSGi Alliance, Tech. Rep., Apr. 2018.

[6] https://docs.oracle.com/middleware/1212/wls/WLPRG/overview.htm#WLPRG107.

[7] G. Di Modica, F. Pantano, and O. Tomarchio, "Snps: An osgi-based middleware for wireless sensor networks," 09 2013, pp. 1–12.

[8] https://projects.eclipse.org/proposals/eclipse-sensinact.

[9] https://www.w3.org/TR/wot-architecture/.

[10] "Community metrics report," http://download.ros.org/downloads/metrics/metrics-report-2019-07.pdf, July 2019.

[11] M. Aragão, P. Moreno, and A. Bernardino, "Middleware interoperability for robotics: A ros–yarp framework," *Frontiers Robotics AI*, vol. 3, p. 64, 2016.

[12] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, 03 2006.

[13] R. Arrais, P. Ribeiro, H. Domingos, and G. Veiga, "Robin: An open-source middleware for plug'n'produce of cyber-physical systems," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 1729881420910316, 2020. [Online]. Available: https://doi.org/10.1177/1729881420910316

[14] A. Pletsch, "Codesys eases programming for multiple controls hardware," vol. 50, pp. 34–35, 06 2004.

[15] A. Santos, A. Cunha, N. Macedo, R. Arrais, and F. N. dos Santos, "Mining the usage patterns of ros primitives," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3855–3860.

[16] N. Mohamed, J. Al-Jaroodi, S. Lazarova-Molnar, and I. Jawhar, "Middleware challenges for cyber-physical systems," *Scalable Computing. Practice and Experience*, vol. 18, no. 4, pp. 331–346, 2017.

[17] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the internet of things," pp. 21–26, 2012.

[18] N. Rosa, D. Cavalcanti, G. Campos, and A. Silva, "Adaptive middleware in go - a software architecture-based approach," *J Internet Serv Appl 11, 3 (2020)*, 05 2020.

[19] T. Tomlinson and J. VanDyk, "Xml-rpc," 01 2010.

[20] R.Nicholson, T.Ward, D.Baum, X.Tao, D.Conzon, and E.Ferrera, "Dynamic fog computing platform for event-driven deployment and orchestration of distributed internet of things applications," pp. 239–246, 2019.