An ontology-based approach to improve the accessibility of ROS-based robotic systems

Ilaria Tiddi
Knowledge Media Institute
The Open University
United Kingdom
ilaria.tiddi@open.ac.uk

Emanuele Bastianelli
Knowledge Media Institute
The Open University
United Kingdom
emanuele.bastianelli@open.ac.uk

Gianluca Bardaro
AI and Robotics Laboratory
Politecnico di Milano
Italy
gianluca.bardaro@polimi.it

Mathieu d'Aquin
Insight Centre for Data Analytics
National University of Ireland,
Galway
mathieu.daquin@insight-centre.org

Enrico Motta
Knowledge Media Institute
The Open University
United Kingdom
enrico.motta@open.ac.uk

ABSTRACT

The focus of this work is to exploit ontologies to make robotic systems more accessible to non-expert users, therefore supporting the deployment of robot-integrated applications. Due to the increasing number of robotic platforms available for commercial use, robotic systems are nowadays being approached by users with different backgrounds, who are often more interested in the robots' high-level capabilities than their technical architecture. Without the right expertise however, using robots is restricted to the capabilities exposed by the platform provider, i.e. they can only be used as end products rather than as development platforms. Our hypothesis is that an ontological representation of the capabilities of robots could make these capabilities more accessible, reducing the complexity of robot programming and enabling non-experts to exploit these systems to a much larger extent. To demonstrate this, an ontology abstracting the capabilities exposed by the most common robotic middleware (ROS) is integrated in a system to allow non-experts to program robots of different types and capabilities without previous knowledge either of the specific robotic platform being considered, or of the intricate systems used in its implementation. Our experiments, in which non-experts users had to configure the system in order to make robots achieve different tasks, show how the efforts required for realizing basic tasks using available robotic platforms can be sensibly reduced through our approach.

CCS CONCEPTS

• **Computing methodologies** → *Knowledge representation and reasoning*; **Mobile agents**; • **Computer systems organization** → *External interfaces for robotics*;

K-CAP 2017, December 4–6, 2017, Austin, TX, USA

ACM ISBN 978-1-4503-5553-7/17/12…$15.00
https://doi.org/10.1145/3148011.3148014

KEYWORDS

Artificial Intelligence, Knowledge Representation, Robotics

ACM Reference Format:
Ilaria Tiddi, Emanuele Bastianelli, Gianluca Bardaro, Mathieu d'Aquin, and Enrico Motta. 2017. An ontology-based approach to improve the accessibility of ROS-based robotic systems. In *Proceedings of K-CAP 2017: Knowledge Capture Conference (K-CAP 2017).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3148011.3148014

1 INTRODUCTION

Autonomous mobile agents and robotics in general are experiencing a new wave of interest thanks to the accelerating advancements in many areas, such as computer vision, artificial intelligence, data management and communication, sensors, and actuators [19]. Today, a considerable amount of efficient techniques for basic robotic tasks (perception, manipulation, navigation etc.), as well as hardware and software components, are available, along with an increasing number of cost-accessible robotic platforms in the market [23].

As a consequence, robotic systems are being approached by users with different backgrounds, who are often less interested in the *low-level* technological components making up the system (e.g. communication, synchronization, drivers) [22], than they are in exploiting the capabilities offered by the robot at a higher level (e.g. autonomous navigation, vision, natural language generation). Lacking the required technical knowledge to exploit such capabilities for their own purpose, these users are then restricted to using the system as an end-product, within the limits of usage anticipated by the robot's provider (e.g. remote control interfaces, and in some cases, restricted programming facilities/APIs).

We argue that this is a waste of the potential of current robotic platforms. Indeed, as support for smart cities for example, robots could be integrated in a number of applications such as parking monitoring, building surveillance or garbage collection. Without the expertise of dedicated robot developers, however, the implementation of such scenarios is only limited to using the capabilities allowed by the commercial platforms (e.g. using a remote-controlled drone to record photos or videos), while more advanced usages,

supported in principle by those systems, are not actually achievable (e.g. programming the drone to autonomously survey an area through several of its sensors).

1.1 Robotic Middleware

The limitations described above could be overcome by integrating a middleware between the application and the robots, able to abstract from the technical implementation of the robotic platform, to enable access and exploitation of the robot's high-level capabilities (vision, movement, perception, actuation) in a simplified and homogeneous manner.

A step towards this is the ROS framework (Robot Operating System).[1] ROS is a collaborative project developed with the aim to relieve developers from the management of low-level components. It has been considerably promoted by the robotics community in recent years, with many current robotic platforms (including commercial, low cost ones such as drones) having been developed directly with ROS, or having been made compliant afterwards.

ROS however remains a low-level technical platform, providing a layer directly above the hardware components of a robot and enabling expert developers to more easily share and reuse specific modules. In other words, programming robots for high-level tasks with ROS still requires a fine-grained understanding of the technical architecture of the robot, and advanced knowledge of robotics, computer programming, and of the ROS framework itself. This particular issue is further emphasized considering that, while there are only a limited number of different types of high-level capabilities a robot might offer, there is a high variety of different low-level components to actually implement them.

In this paper, we focus on simplifying the time-consuming process of configuring/programming a robot for specific tasks, and in enabling it for non-expert users of robotic platforms. Our hypothesis is that this can be achieved through using an additional ontological representation of the high-level capabilities offered by a robotic platform on top of the existing ROS middleware, as an abstraction and an intermediary to the actual technical realization of those capabilities within the system.

1.2 Motivating scenario: Integrating robots in smart-cities

Milton Keynes is a city in Buckinghamshire, England, growing in attention not only for being an example of modern urban design, but also for being the fastest growing city in the UK (in terms of jobs, people and houses). The city engaged in a large "Future City" program, at the center of which the MK:Smart project[2], which has developed a state-of-the-art data acquisition and management infrastructure (the MK Data Hub[3]) and an IoT network with live sensors capturing many aspects of the functionalities of the city (energy and water consumption, transport data, satellite-acquired data, social and economic datasets, and crowdsourced data from social media or specialized applications). The MK Data Hub was built with the idea that a common facility to efficiently manage, integrate and re-deliver data from a variety of sources could be exploited by applications and services, reducing their development costs and enabling intelligent data management (mining, analytics, aggregation, alignment, linking) at the scale of the entire city.

Our current goal is to create applications where robots are also integrated, namely by developing scenarios where robots act both as data collectors [28] and data consumers [9] of the Data Hub. In a practical example, we are looking into exploiting teams of robots of different characteristics and capabilities (i.e. drones and ground robots) for the surveillance and maintenance of green spaces. The main problem we encountered here is that, without the required expertise especially in application development in ROS, we can only exploit the few capabilities exposed by the drone interface (namely, tele-operated navigation and video recording), while more advanced abilities could have enabled our application to better exploit the drone (e.g. through programmed trajectories, or the ability to use the drone's sensors for surveying the area or even to perform object detection with ARtags[4]). Even with the required expertise in robotic application development and a sufficient understanding of the drone's technical architecture, the task of implementing an application accessing such capabilities at a low-level would still represent a major effort, even more so if needing to integrate several different robot architectures (different types of drones and ground robots).

1.3 Proposed approach and contributions

From the scenario presented above, it is clear that the problem to tackle relates to the ability to program different robotic platforms to their full extent homogeneously and with reduced effort, i.e. without incurring into the time-expensive process of learning low-level ROS programming for the specific set of robot architectures at hand. Ontological representations have been used both as a mean to improve system interoperability and to provide meaningful, conceptual abstractions of complex and detailed domains. The question arising here is therefore: *Could an ontological representation of robot capabilities, able to abstract from the low-level implementation of the robot, improve the ability of non-experts to exploit such a robot to achieve specific tasks?* It can be expected that answering this question positively would provide a way to facilitate the integration of robots in a larger variety of applications and environments, including smart-cities.

To answer this question, we propose to develop a system able to understand the capabilities of a robot by relying on an ontology that abstracts from the capabilities of ROS components, and then to use such a system to show how non-experts can access and instruct robots of different types and capabilities without previous experience in doing so. The main idea here is to abstract the ROS layer using an ontological representation, where the ROS components running on the robot are mapped onto capabilities of a higher level. For example, given a ROS component connected to the robot motor and producing sensor data such as velocity or acceleration, we can derive that the robot is able to manage its speed, hence possessing the capability of Movement.

[1] http://www.ros.org/
[2] http://www.mksmart.org/
[3] https://datahub.mksmart.org/

[4] Markers supporting augmented reality tasks such as, the appearance of virtual objects and video tracking to calculate a camera's position, see http://www.hitl.washington.edu/artoolkit/.

To achieve this, the following steps were carried out: (i) studying and understanding the ROS middleware, in order to be able to abstract the capabilities of ROS components; (ii) formalizing the ROS framework into an ontology, which generalizes from specific robotic platforms; and (iii) enclosing the ontology in a system to automatically understand a robot's capabilities.

In summary, the main contributions of this paper are:

- A method to extract an ontological representation of a robot's capabilities from its ROS configuration.
- A tool using this ontological representation as an intermediary between the robot's architecture and a programming interface using a set of common, homogeneous instructions in a simple visual language.
- A demonstration that this tool enables non-expert users to use robots to achieve specific, non-trivial tasks.
- Showing that this tool allows generalization from multiple types of robotic platforms, supporting interoperability.

2 RELATED WORK

This section focuses first on how ontologies were used to facilitate the use of complex systems, then on how capabilities were represented in robotics.

Our work very much relates to the area of *sensemaking*, where meaningful representations are used to facilitate insight and subsequent intelligent actions [25]. External knowledge in the form of ontologies has been used to achieve a wide range of tasks, such as semantic enrichment [2, 16], visualization [4, 30], text processing [1, 29], and enterprise knowledge management [18, 24].

The power of ontologies was also promoted in the Internet-of-Things (IoT), namely by initiatives such as the Semantic Sensor Web for networks and sensors [8, 26], smart homes and environments integrating heterogeneous devices and services and allowing personalized interactions [7, 31], as well as smart products [10, 20]. Similar to our work, the main objective here is to use semantic technologies to enable the abstraction of the heterogeneous technologies such as sensors and devices, so to improve interoperability and simplicity, support non-expert end-users and allow the development of more situation-aware applications. These works have generally focused either on representing the information related to the sensing devices, without trying to abstract actual capabilities, or on the development of middleware frameworks, therefore focusing on the data-integration problem. An attempt to combine these two aspects is presented in [15].

In robotics, the need for standards and common vocabularies to promote interoperability and reusability has been raised by the Ontologies for Robotics and Automation working group[5] (ORA) sponsored by the IEEE Robotics & Automation Society, which presented the CORA ontology to describe the most general concepts and relationships in robotics and automation [11]. An ontology to define devices with widely differing capabilities, software requirements and communication technologies is integrated in the middleware presented by [13]. A canonical robotic command language (CRCL) was presented by [3] to provide generic commands which are not specific to the language of either the planning system or the robot controller. With respect to our work, these representations focus on low-level functional aspects of the robot as a device, e.g. its parts, positions and coordinates, without leaving any possibility of abstraction.

Formal models for robot capabilities can be found in the area of task allocation and planning. This body of works defines capabilities as related to some kind of resource, e.g., sensors/actuators [17], processing capacities or software modules [6, 21]. Others define capabilities as subtasks or basic skills [12, 14]. A task-independent model of robot capabilities is also presented by [5]. In [27], representations of robot actions and perceptions are combined with common-sense knowledge in the context of autonomous robot control. These models differ from our work in that capabilities do not abstract from the hardware/software architectures, i.e. they represent *how* a robot can do something (and *what*) at a fine-grained or medium level of granularity. Also, tasks are generally domain-specific and manually defined.

Finally, robots with higher level capabilities have been used in systems aiming at teaching computer programming, such as Dash[6] or the Blocky[7] visual programming tool. These tools are however only focused on the educational aspects, through interactivity and advanced visualization.

3 AN ONTOLOGY-BASED SYSTEM TO ABSTRACT ROBOT CAPABILITIES

In this section, we present the ontology-based system which extracts and abstracts the capabilities of robotic platforms. We start with a more detailed description of the ROS framework.

3.1 ROS: The Robot Operating System

ROS is a collection of tools, libraries, and conventions that aim at simplifying the creation of complex and robust robot behaviors across a wide variety of robotic platforms. The main idea behind it is to free developers as much as possible from the burden of managing the communication between components, as well as promoting the decomposition of their functionalities.

The core of any ROS application is the *computational graph*, which consists of a network of all the data processes involved. Main elements of this graph are *nodes*, *messages*, *topics* and *services*, described in the following.

- *Nodes* consist of sets of processes performing a number of computations. They are the minimal executable unit of ROS and each one implements a specific functionality. For example, the `move_base` node implements the robot navigation, `kobuki_node` is a wrapper for the motors' driver, and `map_server` is a node broadcasting the map of an environment to the entire system.
- *Messages* consist of the typed data structures exchanged by the nodes when they communicate with each other. An example is the `Twist` message, composed by six fields (three linear and three angular velocities), used by the `move_base` node to send setpoints to `kobuki_node` controlling the wheels.
- *Topics* are used to route messages between nodes asynchronously with a publish/subscribe paradigm. Generally speaking, a sending node creates a message and it publishes it on a topic, and the

[5] https://standards.ieee.org/develop/wg/ORA.html

[6] https://uk.makewonder.com/dash

[7] http://wiki.ros.org/blockly

nodes that are interested in that message will subscribe to it. For instance, `move_base` and `kobuki_node` exchange `Twist` messages using a topic called `/cmd_vel`. Note that multiple nodes can publish on or subscribe to the same topic, but neither producers nor consumers know who is producing/receiving the information. The topic is therefore only a named channel upon which nodes agree, and not a direct connection between them.

- *Services* are communications channels to route messages synchronously through a request/reply (or client/server) paradigm. They are defined by a pair of messages, one for the request and one for the response. A server node provides a service and then waits for a request, while a client node requires a service and wait for the server to provide a response. An example of a service is the one used by `move_base` to request the map from the `map_server` node. Since the exchange of the map message `OccupancyGrid` is too memory-demanding to be constantly published on a topic, `move_base` requests the map only once, synchronously, through the `/map` service, and then stores it locally.

Additional ROS elements also play a role in the computational graph. Being of less relevance to the purpose of this work, they have been omitted by our current implementation.

3.2 Abstracting capabilities from ROS

Once understood the main principles of ROS, the second step is to provide a formal representation of ROS and its components. It is worth mentioning here that, although our ROS representation is inspired by the Event[8] and Situation[9] ontology design patterns, we use the ontology as a mean to demonstrate our hypothesis, and it is therefore out of the scope of this work to assess its originality or robustness.

ROS Communication. The main ROS concepts presented above are represented as a `ros:ROSCommunication`, depicted in Figure 1(a).

As one can see, a `ros:ROSCommunication` consists in a `ros:Message` routed via a `ros:Mode` (consisting in a topic or a service communication) and a set of `ros:CommunicationComponents` (publishers, subscribers, clients and servers), that instances of a `ros:Node` use as a mean to communicate with each other. The class `ros:Package` is used to identify the library to which the message belongs, since these are generally named in the form of `/$package/$message`. For example, the `Twist` message published by `move_base` belongs to the package `geometry_msgs`.

The publish/subscribe paradigm is represented as a class `ros:ROSTopicCommunication` in Figure 1(b). Here, messages are communicated through instances of `ros:Publisher` and `ros:Subscriber` via instances of a specific `ROS:Topic`. Taking back the example of `move_base` and `kobuki_node` of Section 3.1, we might have the following basic representation:

```
@prefix ros: <http://data.mksmart.org/onto-ros/class#>
@prefix : <http://data.mksmart.org/onto-ros/resource/>
:example1 a ros:TopicCommunication;
       ros:isRoutedVia :cmd-vel;
       ros:hasMessage :twist;
       ros:hasPublisher :publisher1;
       ros:hasSubscriber :subscriber1.
:twist a ros:Message.
:cmd-vel a ros:Topic.
:move-base a ros:Node;
       ros:hasComponent :publisher1.
:kobuki-node a ros:Node;
       ros:hasComponent :subscriber1.
```

A `ros:ServiceCommunication` is represented in Figure 1(c) as composed by the pair ⟨`ros:RequestMessage`,`ros:ResponseMessage`⟩, respectively sent by instances of a `ros:ServiceClient` and a `ros:ServiceServer`.

The service is offered through an instance of the `ros:Service` class, which specifies the `ros:Mode` of Figure 1(a). A minimal representation of a service exchanging the map between `move_base` and the `map_server` node would, for instance, be as follows:

```
@prefix ros: <http://data.mksmart.org/onto-ros/class#>
@prefix : <http://data.mksmart.org/onto-ros/resource/>
:example2 a ros:ServiceCommunication;
       ros:isRoutedVia :map;
       ros:hasMessage :occupancy-grid;
       ros:hasClient :client1;
       ros:hasServer :server1.
:occupancy-grid a ros:Message.
:map a ros:Service.
:move-base a ros:Node;
       ros:hasComponent :client1.
:map-server a ros:Node;
       ros:hasComponent :server1.
```

Components capabilities. In Figure 2 we show how ROS components are mapped to `ros:Capabilities`. The main insight here is that high-level capabilities are implemented in the robot through a set of ROS nodes, messages, topics and services, which we can map and represent in a knowledge base to infer what the robot is able to do.

Let us consider again the example of `move_base` that publishes the `Twist` messages on the `/cmd_vel` topic. By analyzing the computational graph while the robot is operating, one can notice that a communication component, published by `move_base`, is producing setpoints on `/cmd_vel`, and therefore derive that the robot is able to move, hence owning a capability such as `capa:Directional_Movement`.

Based on this example, establishing the mappings between a ROS system and the capabilities offered by the corresponding robot (using the schema of Figure 2) is equivalent to creating a set of rules whose premises are specific sets of ⟨`ros:Node`,`ros:Message`,`ros:Topic`⟩, which allow to derive the presence of a capability. Unfortunately, ROS does not provide standards for naming topics or nodes. Developers implement their own nodes and can choose a different name for the topic where the setpoints will be published. This lack of standardization makes it difficult to rely on topics and nodes to discriminate the robot capabilities in the ROS computational graph. However, a major effort has been put into standardizing ROS messages, and a wide variety of messages are now commonly used to represent most of the information used by ROS-based systems.[10] Messages have a clear semantics: For instance, `geometry_msgs/Pose` will only provide information about the position of the robot in a

[8] http://krisnadhi.github.io/onto/event.owl
[9] http://www.ontologydesignpatterns.org/cp/owl/situation.owl
[10] http://wiki.ros.org/common_msgs

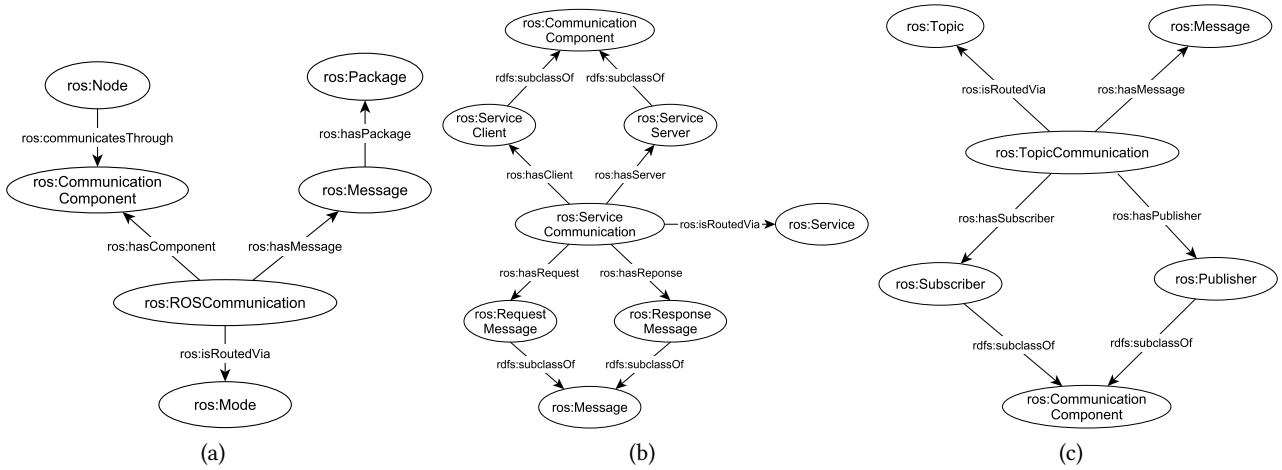

Figure 1: ROS basic structure (a), Structure of `ros:Topic` (b) and Structure of a `ros:Service` (c).

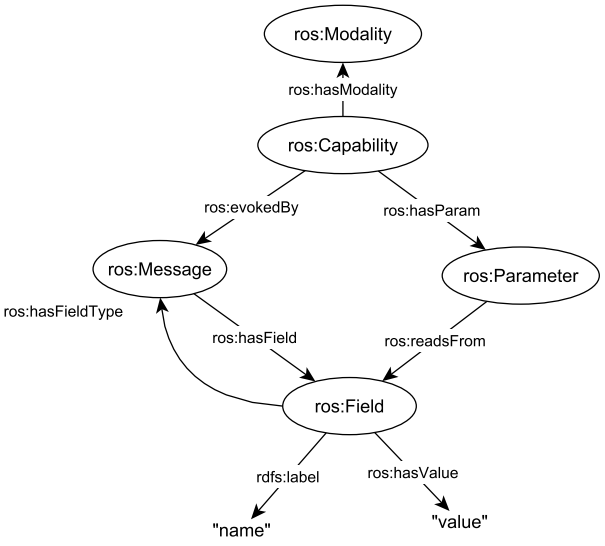

Figure 2: Mapping ROS components to capabilities.

3D space. We can therefore encode this convention as a triple (rule) of the form ⟨`ros:Message,ros:evokes,ros:Capability`⟩.

In addition, the way messages are exchanged can also be exploited to identify capabilities, e.g. a publisher of a `Pose` message identifies the current position of the robot, therefore evoking the ability of sensing itself (self perception), while a subscriber of `Pose` expects a desired position to be send to the robot, evoking the ability of autonomous navigation. In order to represent how different capabilities can be evoked by a read or a published message, we use triples of the form ⟨`ros:Capability,ros:hasModality,ros:Modality`⟩. A modality is either read or write, representing respectively capabilities giving information about the robot (e.g. sensor data) and capabilities expecting some inputs (e.g. navigation). In the remainder of this paper, we will refer to these as *read capabilities* and *write capabilities*.

Finally, a `ros:Capability` might also use the message fields to identify how to parametrize the capability to achieve the desired behavior of the robot, in the case of a write capability, or which is the information carried by a message correlated to the specific capability.

Capability taxonomy. Based on the principles outlined above, we designed a basic taxonomy[11] including the mappings between ROS components (especially messages) and specific capabilities. These were defined in a bottom-up fashion, by collecting the specific capabilities from the robotic platforms supported by ROS[12] as well as from the most adopted ROS libraries, then abstracting them iteratively onto capabilities of higher levels. Due to space limitations, we only present here the three macro-classes we used in our work: `Sensing`, which includes all the capabilities enabled by the robot sensors (vision, depth sensing, light sensing etc.); `Movement`, comprising the activities related to changing position (tele-operation, body-part movement, autonomous navigation etc.); and *RobotKnowledge*, which includes capabilities the robots might have to represent their surrounding environment (such as the map representation). While establishing the taxonomy of capabilities is naturally a constantly evolving work, we asked ROS experts to verify that our current coverage is correct and reasonably complete.

3.3 Extracting capabilities from ROS

The final step consists of using the defined ontology to create a system that can automatically extract high level capabilities by inspecting the ROS setup of a robot. The system's architecture, shown in Figure 3, relies on the following components:

1) The *Robot*, representing the architecture of the physical system. It consists of a collection of ROS nodes and topics currently in execution on a specific platform.
2) The *Dynamic Node*, which dynamically creates publishers and subscribers using the information derived from the ontology.

[11] Available at https://tinyurl.com/ybqg7om5. Being an evolving work, further contributions to the taxonomy are left for future work.
[12] http://robots.ros.org/

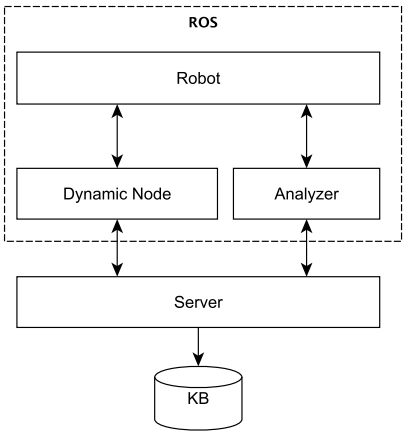

Figure 3: System general architecture.

In the case of a request for a reading capability, the dynamic node creates a subscriber associated to the specific topic, and relays to the server the messages received from the robot. For writing capabilities, the parameters received from the server are structured in a message and then sent to the platform through an ad-hoc publisher.

3) The *Analyzer*, interfacing with the robot system to determine all the capabilities the robot processes. This component scans all the active topics in the robot system, and then associates to the relevant topics an appropriate capability based on the ontology based on the mappings described previously.
4) The *Server*, acting as a bridge between the robot system implemented in ROS and the outer world (namely, the users). The server invokes the analyzer when starting, in order to populate the knowledge base with the available capabilities and topics, and translates the high level capabilities to the correct ROS topics every time it receives a request.

4 EVALUATION

We set up our evaluation with the goal of showing how our ontology-based approach could indeed make robots more accessible to users without previous expertise in robotics or ROS. To this end, we designed a user-interface wrapping our system, and asked non-experts users to use this interface to instruct robots of different types to solve some high-level tasks. We measured indicators of the efforts required to achieve those tasks, as described below after an introduction to the interface.

4.1 User interface

To make robots' capabilities available to users, we created a basic *imperative* programming language, in which the atomic blocks are invocations of the available robot capabilities (e.g. navigating to a certain coordinate). We included in this language also basic programming constructs, such as *if-then-else*, *while-do* and *repeat* statements, allowing the user to build condition-constrained behaviors. The parameters of a capability could also be used in the conditions, so to exploit any robot output to drive the program flow (e.g. moving forward until an ARtag is detected). Such a language could be easily extended with common features of other existing languages (*for* loops, *break* and the use of variables); however, we considered this set of programming blocks sufficient for the purpose of our experiment.

Given a robot running on ROS, the user was shown an interface, as in Figure 4, including: (i) a left pane showing the robot capabilities as inferred by the ontology-based system, as well as the necessary control parameters; (ii) a right pane with the list of available programming blocks (right side) and (iii) a central pane for building the robot program, i.e. a sequence of blocks to be executed.

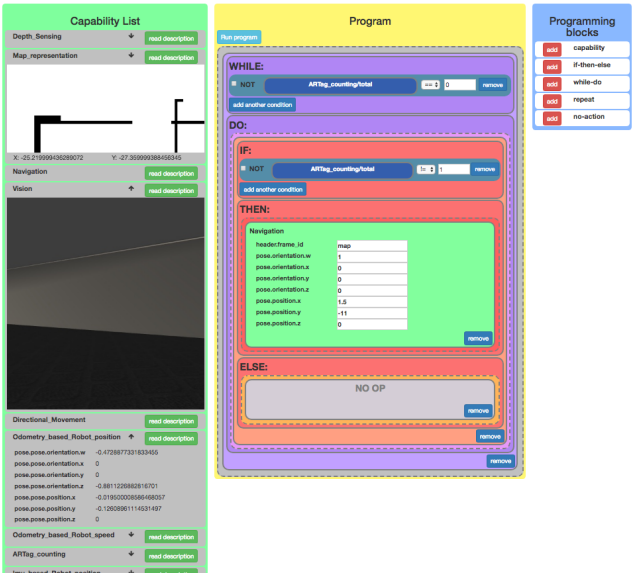

Figure 4: Interface presented to the users in order to program a robot. Available capabilities are on the left, the possible programming blocks to be used on the right, and the main program is in the middle.

4.2 Experimental Setup

Four exercises of increasing difficulty were asked to be performed by each user, which corresponded to creating a program allowing a robot to achieve a specific task. In order to demonstrate that the ontology-based system could allow the abstraction of robot capabilities independently from the platform, we set up two variants of each exercise, a simulated one with a ground wheeled robot operating in an office environment (`s`-variant), and a real-world one with a drone flying in an indoor space (`r`-variant). Table 1 presents the robot capabilities available in each setting[13].

Exercise 1: Single command. The first exercise required to implement a single movement behavior. In the `s`-variant, the user had to instruct the robot to move to a specific point in space corresponding to a room. Similarly, in the `r`-variant, the user had to instruct the drone with a single movement command.

[13]For clarity, autonomous navigation is related to the action "go to a coordinate X,Y", while directional movement is intended as commands such as "turn right" or "move forward".

Table 1: Robot capabilities for the two exercise variants.

Mode	s-variant	r-variant
write	autonomous navigation directional movement	take-off land directional movement
read	vision current position current speed map representation object recognition	vision current position object recognition

Exercise 2: Command sequence. In the second exercise, users needed to instruct a robot with a sequence of two single actions. In the s-variant, the users needed to instruct the robot to navigate to two rooms of the office, one after the other. In the r-variant, the user had to instruct the drone with any motion command, then ending its movement with a landing command.

Exercise 3: Condition-based halt. The third exercise required users to implement a sequence of actions with a termination condition. In the s-variant, the robot needed to perform a patrolling of three rooms of the office, stopping only once all the rooms had been visited at least twice. In the r-variant, the user had to instruct the drone to keep on turning on itself until an ARtag was seen, in which case it would land.

Exercise 4: Object recognition. In the final exercise, users had to instruct the robot to perform autonomous navigation through several ARtags. In the s-variant, the robot had to patrol a set of rooms until an ARtag was detected. In the r-variant, the user had to implement a behavior for the drone to perform 3 different movement actions every time an ARtag was seen.

4.3 Results and discussion

A total of 14 users were involved in the evaluation, equally shared between the s- and the r-variant. Those users were essentially members of our research lab with at least some basic programming skills (to make the exercise meaningful), but with no experience in either ROS or robotics in general. As a starting point, users were first allowed to familiarize themselves with the interface, namely through clicking on the different sections to understand the general behavior of the tool. Users were however prevented from reading the description of the capabilities. After this first step, they were asked to solve all four exercises one after the other. For every exercise, we measured the time from the end of the task description until the final execution of the program.

Table 2 shows the average time *avg*(T) required by the users to solve each exercise, along with the average number of programming blocks *avg*(PB) and the number of capabilities *num*(CAP) required to solve the task.

All the exercises were successfully carried out by all users. As one can see, Ex. 1 took slightly longer (especially in the s-variant), when compared with other more complex exercises. This can be attributed to the time users required to familiarize themselves with the capabilities of the robot they were working with, which they could not have known beforehand. The relatively high variance in the time taken for Ex. 4 is due to this particular exercise having multiple solutions, some of which taking longer to implement than others.

Table 2: Results for the s-variant and the r-variant. These are compared with the efforts required by a ROS expert to achieve the same tasks, measured in terms of number of lines of code *num*(LN), ROS components *num*(COM) and ROS message types *num*(MSG) that were employed.

		Ex. 1	Ex. 2	Ex. 3	Ex. 4
s-variant					
users	*avg*(PB)	1	2	4	9.5
	num(CAP)	1	1	1	2
	avg(T)	1:22 ± 42s	1:04 ± 23s	1:15 ± 16s	6:52 ± 1:46
expert	*num*(LN)	35	58	64	82
	num(COM)	1	2	2	3
	num(MSG)	1	2	2	3
r-variant					
users	*avg*(PB)	1	2	4	8
	num(CAP)	1	2	4	4
	avg(T)	1:16 ± 3s	01:16 ± 8s	4:05 ± 15s	5:47 ± 1:39
expert	*num*(LN)	34	39	56	59
	num(COM)	1	2	4	4
	num(MSG)	1	2	3	3

A key, straightforward conclusion from this table is that users of this tool, who had no experience of programming robots and no prior knowledge of the architecture of the robot they were manipulating, managed to successfully program such a robot to achieve tasks from the very simple Ex. 1 to the more difficult Ex. 4 in a matter of a few minutes. Considering the inherent complexity of robot programming and of understanding not only what a robot can do (what capabilities it possesses), but also how to use it (how to invoke those capabilities), this can be considered a non-trivial achievement.

A direct comparison with how the same users would have achieved the same tasks without the tool provided is not feasible and would turn out to be meaningless. However, it appears a straightforward assumption that, those users not being familiar with ROS, the simple (in our tool) process to understand the different components of the robot, what they do and, crucially, how to invoke them, would require more than a few minutes by itself. ROS is a complex framework, requiring hours of practice to master. In addition, analyzing the computational graph of the specific robot to understand which topics and services are being be used (i.e. what the tool does through the ontology) is far from an easy task. A number of ROS nodes would need to be implemented from scratch to encapsulate the required functionalities, and managing the correct publishers and subscribers. Lastly, the nodes would need to be deployed and integrated with the robot architecture. Knowledge of specific packages (e.g. the `move_base` node for autonomous navigation) is also required by some of the exercises. In other words, while a direct comparison could not have been performed, there is little doubt that significantly more effort would have been required to enable our non-expert users to achieve the same results with ROS, as it did with our tool.

As an additional point towards the validity of our claim that our tool reduces the effort required to exploit robots' capabilities and therefore make them more accessible, we asked an expert in

robotics with extensive experience in ROS to achieve the same task. Once again, the objective here is not to compare the experts to the non-experts using two different frameworks, but to provide an intuitive understanding of the difficulty of realizing the tasks achieved by our users without our tool. In Table 2, we therefore show for each task in each variant:

- The number *num*(LN) of lines of code used by the ROS expert,
- The number *num*(COM) of ROS communication components (publishers and subscribers) employed
- The number *num*(MSG) of message types used.

These metrics give an estimate of the effort required by a ROS developer to solve the specified tasks. Lines of code set a lower bound for the implementation time, while number of components and messages outline the complexity of the solution.

This comparison further show how programming a robot is made "easier" and, through abstracting capabilities from the technical aspects of their implementation, requires less complexity. Our approach and the associated tool therefore represent a viable solution to enable non-expert users to exploit robots in ways that were before only accessible to expert ROS programmers.

5 CONCLUSIONS

In this paper, we have shown how a layer of ontological knowledge can empower non-expert users to access robotic systems of different types and capabilities. We developed an ontology-based system for robot programming abstracting from the specific components of the robot operating system (ROS), and showed how this allows non-experts to make robots achieve specific tasks without having any previous experience in robotics.

Several directions can be taken as future work. We are looking into refining the taxonomy of robot capabilities, to allow both high-level and fine-grained actions. In addition, we are also interested in testing our system with other types of robots, e.g. by employing robots with manipulators. Another possible direction is to extend the system into a collaborative programming environment were multiple users can work together in programming a (set of) robots. Finally, we intend to make the system reusable by providing the high-level capabilities in the form of public APIs.

REFERENCES

[1] G Aguado, A Bañón, J Bateman, S Bernardos, M Fernández, A Gómez-Pérez, E Nieto, A Olalla, R Plaza, and A Sánchez. 1998. ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In *Workshop on Applications of Ontologies and Problem Solving Methods, ECAI*, Vol. 98.

[2] Sören Auer, Sebastian Dietzold, and Thomas Riechert. 2006. Ontowiki-a tool for social, semantic collaboration. In *International Semantic Web Conference*, Vol. 4273. Springer, 736–749.

[3] Stephen Balakirsky, Thomas Kramer, Zeid Kootbally, and Anthony Pietromartire. 2012. Metrics and test methods for industrial kit building. *Gaithersburg, MD2013* (2012).

[4] Marius Brade, Florian Schneider, Angelika Salmen, and Rainer Groh. 2013. OntoSketch: Towards digital sketching as a tool for creating and extending ontologies for non-experts. In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*. ACM, 9.

[5] Jennifer Elisabeth Buehler and Maurice Pagnucco. 2014. A Framework for Task Planning in Heterogeneous Multi Robot Systems Based on Robot Capabilities.. In *AAAI*. 2527–2533.

[6] Jian Chen and Dong Sun. 2010. An online coalition based approach to solving resource constrained multirobot task allocation problem. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*. IEEE, 92–97.

[7] Liming Chen, Chris Nugent, Maurice Mulvenna, Dewar Finlay, and Xin Hong. 2009. Semantic smart homes: towards knowledge rich assisted living environments. *Intelligent Patient Management* (2009), 279–296.

[8] Oscar Corcho and Raúl García-Castro. 2010. Five challenges for the semantic sensor web. *Semantic Web* 1, 1, 2 (2010), 121–125.

[9] Enrico Daga, Mathieu d'Aquin, Alessandro Adamou, and Enrico Motta. 2016. Addressing exploitability of smart city data. In *Smart Cities Conference (ISC2), 2016 IEEE International*. IEEE, 1–6.

[10] Mathieu d'Aquin, Enrico Motta, Andriy Nikolov, and Keerthi Thomas. 2012. Realizing Networks of Proactive Smart Products. In *Knowledge Engineering and Knowledge Management*. Vol. 7603. Springer Belrin Heidelberg, 333–352.

[11] Sandro Rama Fiorini, Joel Luis Carbonera, Paulo Gonçalves, Vitor AM Jorge, Vítor Fortes Rey, Tamás Haidegger, Mara Abel, Signe A Redfield, Stephen Balakirsky, Veera Ragavan, et al. 2015. Extensions to the core ontology for robotics and automation. *Robotics and Computer-Integrated Manufacturing* 33 (2015), 3–11.

[12] Cheng-Heng Fua and Shuzhi Sam Ge. 2005. COBOS: Cooperative backoff adaptive scheme for multirobot task allocation. *IEEE Transactions on Robotics* 21, 6 (2005), 1168–1178.

[13] Arthur Herzog, Daniel Jacobi, and Alejandro Buchmann. 2008. A3ME-an Agent-Based middleware approach for mixed mode environments. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM'08. The Second International Conference on*. IEEE, 191–196.

[14] Jacob Huckaby and Henrik I Christensen. 2012. A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics. In *Workshops at 26th AAAI conference on artificial intelligence*.

[15] Konstantinos Kotis and Artem Katasonov. 2013. Semantic interoperability on the internet of things: The semantic smart gateway framework. *International Journal of Distributed Systems and Technologies (IJDST)* 4, 3 (2013), 47–69.

[16] Markus Krötzsch, Denny Vrandecic, and Max Völkel. 2006. Semantic mediawiki. In *International semantic web conference*, Vol. 4273. Springer, 935–942.

[17] Lars Kunze, Tobias Roehm, and Michael Beetz. 2011. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 5589–5595.

[18] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. 2003. Ontologies for enterprise knowledge management. *IEEE Intelligent Systems* 18, 2 (2003), 26–33.

[19] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. 2013. *Disruptive technologies: Advances that will transform life, business, and the global economy*. Vol. 180. McKinsey Global Institute San Francisco, CA.

[20] Max Mühlhäuser. 2008. Smart products: An introduction. *Constructing ambient intelligence* (2008), 158–164.

[21] Lynne E Parker and Fang Tang. 2006. Building multirobot coalitions through automated task solution synthesis. *Proc. IEEE* 94, 7 (2006), 1289–1305.

[22] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, 5.

[23] Kanna Rajan and Alessandro Saffiotti. 2017. Towards a science of integrated AI and Robotics. (2017).

[24] Liana Razmerita, Albert Angehrn, and Alexander Maedche. 2003. Ontology-based user modeling for knowledge management systems. *User modeling 2003* (2003), 148–148.

[25] Daniel M Russell, Mark J Stefik, Peter Pirolli, and Stuart K Card. 1993. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. ACM, 269–276.

[26] Amit Sheth, Cory Henson, and Satya S Sahoo. 2008. Semantic sensor web. *IEEE Internet computing* 12, 4 (2008).

[27] Moritz Tenorth and Michael Beetz. 2015. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence* (2015).

[28] Ilaria Tiddi, Enrico Daga, Emanuele Bastianelli, and Mathieu d'Aquin. 2016. Update of time-invalid information in knowledge bases through mobile agents. *Integrating Multiple Knowledge Representation and Reasoning Techniques in Robotics* (2016).

[29] Paola Velardi, Roberto Navigli, Alessandro Cuchiarelli, and R Neri. 2005. Evaluation of OntoLearn, a methodology for automatic learning of domain ontologies. *Ontology Learning from Text: Methods, evaluation and applications* 123 (2005), 92.

[30] Horst Werner, Markus Latzina, and Marius Brade. 2011. Symbik—A new medium for collaborative knowledge-intensive work. In *Proceedings of the international conference on education, informatics, and cybernetics*.

[31] Zhaohui Wu, Qing Wu, Hong Cheng, Gang Pan, Minde Zhao, and Jie Sun. 2007. ScudWare: A semantic and adaptive middleware platform for smart vehicle space. *IEEE Transactions on intelligent transportation systems* 8, 1 (2007), 121–132.