Analysis of Tools Used for Implementation of a Knowledge Base Based on an Ontology for a Service Robot in a Kitchen Environment

Grzegorz Kuduk, Maciej Bekas, Barbara Wąsowska, Piotr Palka

Analysis of Tools Used for Implementation of a Knowledge Base Based on an Ontology for a Service Robot in a Kitchen Environment

Grzegorz Kuduk[0000−0002−8794−5001], Maciej Bekas[0000−0002−2848−1534], Barbara Wąsowska, and Piotr Pałka[0000−0002−0006−363X]

Warsaw University of Technology, Warsaw, Poland
{Grzegorz.Kuduk.stud,Maciej.Bekas.stud,Barbara.Wasowska.stud,Piotr.Palka}@pw.edu.pl

Abstract. The work presents the process of systematic invention in relation to the design of a robot companion component. The component is responsible for knowledge management in the kitchen environment and its automatic use. Based on the TRIZ method, in particular the Contradiction Business Matrix 3.0, the Inventive Principles are listed. Then, an analysis is carried out regarding potential tools for the implementation of inventive principles. The results of the analysis carried out on the KnowRob and Armor tools in terms of: documentation quality, difficulty of installation, usability and performance. The analysis was carried out to determine the superior tool in knowledge processing for robots. KnowRob is a popular tool in this area. Armor is a young and interesting tool with a potential to become widely used. These kinds of tools need to respond quickly and guarantee reliability. For this purpose, installations and configurations of both environments were performed and documented. Then, a set of queries in Prolog and SPARQL were prepared and tested. The ontology used in testing is based on Web Ontology Language (OWL). Our findings indicate that KnowRob is the superior tool in the tested areas.

Keywords: Knowrob · Armor · Owl · Robotics · Ontology

1 Introduction

The subject of this work is the process of systematic invention in relation to the design of a component of the robot companion. The goal of the component is to manage the knowledge that is acquired by a robot and the provision of that knowledge to other components of the robotic system. We assume that the robot is supposed to be used to support the work of an elderly person in the kitchen environment. For this purpose, it must efficiently navigate the concepts and dependencies concerning objects used and activities performed. One of the used solutions [9] is equipping the robotic system with a knowledge database. It allows the robot to efficiently navigate the concepts and relations described by the ontology [17] used. It should understand rules prevailing in the kitchen environment and descriptions of procedures provided to it. Furthermore, it should act as a "companion" for an elderly person, helping them in their daily duties.

The proposed research method is based on two methods. First, is the TRIZ-based analysis of the problem of robot companion systems working in the kitchen environment. Second, is an experimental analysis of a set of IT tools that support ontology management and allow query automation.

2 Systematic invention using TRIZ

This section describes the innovation-oriented TRIZ-based method for solving the problem. According to the aforementioned assumptions, consisting of the fact that the companion robot must efficiently navigate the concepts and dependencies concerning objects used and activities performed, the **specific problem** consists in equipping the robot with a knowledge base from which it will derive domain knowledge.

After having the specific problem formulated, we match it to an **abstract problem**. From the TRIZ toolkit, we select the Contradiction Business Matrix 3.0 [13], which is the revised method developed by Darrell Mann, designed to solve problems in contemporary technological conditions. As the robotic system is in the development phase, only the system features (25-32) are taken into account.

- Improving features:
  - (25) amount of information - the robot should have as much knowledge as possible
  - (28) adaptivity/versatility – the robot should be able to perform different tasks
- Worsening feature:
  - (29) system complexity

After analysing the Contradiction Business Matrix 3.0, four following Inventive (Business) Principles are selected: (15) Dynamics, (25) Self-service, (40) Composite Structures, and (28) Another Sense.

The principle 40 points using composite structures for managing the knowledge in the robotic system. There are specific notations, like OWL (Ontology Web Language) [8] for automatic processing of knowledge. The robotic system should allow self-servicing (principle 25) in the sense that it should add, and modify pieces of knowledge, check dependencies among the concepts (e.g. having the information about the positions of a cup and a table, infer the knowledge that the cup is on the table). According to principle 28 (another sense), the robotic system should be equipped with sensors (cameras, artificial skin) to sense the environment. Principle 15 means that the robotic system should be able to work in a dynamic (changing) environment.

The above analysis points to the need for utilizing the knowledge-based system working together with the robot companion. In the further section, the analysis of known tools for knowledge management is described.

3 Overview of the tools

The tools used for implementing the knowledge database described in cited articles [9,16,20] are badly documented, which can cause severe problems for potential users. This article aims to take a look at the installation and configuration of the KnowRob and ARMOR (A ROS Multi-Onotlogy Reference) frameworks, while examining their difficulty of usage and quality of their documentations, as well as test performance using a set of custom queries.

3.1 Service robot

Creation of autonomous agents was aimed at creating self-sufficient agents cooperating with other agents in a changing environment.

The scope of the ontology in autonomous robots must include the definition of objects, the map of the environment, accessibility and influence factor, actions and tasks, activity and behavior, planning and methodology, abilities, skills, hardware components (Unified Robotics Description Format (URDF) area), programming components (Robot Operating System (ROS) area), software and communications.

3.2 Ontology

The first approaches to create an ontology were made in 1993. The official definition and discussion of the problem took place in 2009 [12]. Ontologies were defined as a logical theory made up of sets of formulas. It is understood as a representation of vocabulary from a given field or a theory of this field that defines objects, properties and relations between objects. The goal of an ontology is to explicitly formalize the domain language so that it can be used and interpreted unambiguously. The resulting formulas are intended to represent the concept of the world as logically as possible. Most often ontologies are based on First-Order Logic (FOL) [14] or Web Ontology Language (OWL). Formulas are constructed using units, classes, relationship functions, and axioms. Units are used to map the objects that the ontology deals with, classes - sets of features for defining units, functions allow to identify and bind units, relations describe connections and dependencies between those units, and axioms are expressions connecting all the above-mentioned elements.

Ontologies can be divided according to the classification of language into strongly informal, semi-informal, semi-formal and strictly formal [21]. The informal ones include those that have no formal semantics associated with them. That includes Resource Descritpion Framework (RDF), Unified Modeling Language (UML) or Business Process Model and Notation (BPMN). However, the following article discusses formal ontologies - those that are related to the formal semantics of the language, because it enables their formal interpretation. Such languages include First Order Logic (FOL) and Web Ontology Language (OWL), which contain clear and comprehensive rules of syntax and semantics.

This group is considered to be one of the most reliable languages in the world of technology.

Ontologies can also be categorized according to the scope they cover. We distinguish high-level, reference, domain and application ontologies [15]. Higher-level ontologies describe a wide range of the world, such as Suggested Upper Merged Ontology (SUMO) [5] which covers descriptions of objects, events, high-level relationships, states, belonging, and quality. Reference ontologies focus on a given domain and the description of its components. It is used in the fields of medicine [10], engineering [18] and entertainment [19]. When an ontology focuses on an even more limited scope (e.g. tourism production), it belongs to the domain group. Application ontology deals only with the description of the theory used in a given application, such as CAD/CAM or ERP.

The scope of inference that the robot must perform is recognizing, categorizing, decision-making, perceiving and assessing situations, anticipating and observing, problem solving and planning, as well as reasoning and holding beliefs, performing actions, interacting and communicating, remembering, reflecting and teaching.

Frameworks used in autonomous robots using the knowledge representation approach include for example, KnowRob, IEEE-ORA, ROSETTA, CERESSES [7] and RehabRobo-Onto [11].

3.3 Ontology standardization

Robotic systems need to meet both hardware and functional standards. Many of them result from safety or operational standards established by international organizations. Robot ontology standards such as "Robot Standards and Reference architecture" or "Ontology-based unified robot knowledge for service robots in indoor environments" [20] have also been designed, but they are not widely used.

3.4 ROS

ROS (Robot Operating System) is one of the most widely used middleware in robotics. It transforms software components and communication graphs into nodes, each of which listens to or sends messages and offers a service invoked by other nodes. Messages are described with a defined syntax and generated for a given language syntax, eg Python, C++ by ROS.

3.5 KnowRob

The basic framework used in robotic knowledge databases is the popular KnowRob system. It is a system designed for service robots. The main programming language used in the KnowRob system is SWI Prolog. This language belongs to the First-Order Logic group of languages and includes a library for managing RDF (Resource Description Framework) tuples. In the form of these tuples, the KnowRob system represents knowledge written as facts in an OWL ontology.

The next generation of KnowRob - KnowRob2 - focuses on simulation, rendering techniques and a hybrid knowledge processing architectures.

The analysis and understanding of knowledge consists in collecting the transmitted facts and data and then integrating them. It uses virtual knowledge databases, thanks to which it processes information on the basis of knowledge structures and relations, the processing of which is defined in the adopted ontology.

One of the flaws of the KnowRob system is the shallow representation of the symbolic principle of the behavioral approach to robots. Another much bigger disadvantage is the lack of the proposed representative standards and, although it is one of the most widely used systems, the user community is very small, so without sufficient documentation and with emerging problems, each user relies on their own knowledge or searching untested sources.

3.6 ARMOR

Armor is a framework for managing one or more ontologies within ROS. It allows users to use an ontology in robotics without knowing Java, which is needed to use AMOR.

4 Documentation

4.1 KnowRob

Framework KnowRob has documentation available in two different places: on its own website[3] and on the github repository site[4]. Documentation which is placed on the github repository site has an advantage in the form of its topicality - it is always compatible with the available framework version. However, this documentation contains very limited information and does not provide descriptions of functions' advanced usage. Framework elements widely described on KnowRob's page are mostly deprecated and do not apply to newer versions. Information might be partially usable, but its proper functionality is not guaranteed. For example, function owl_parse\1 which is included in documentation is not available in the latest version.

4.2 ARMOR

The ARMOR framework's disadvantage is its documentation. ARMOR does not have a documentation aside from the one posted on the github repository. It is very limited and contains little information beyond basic usage of the methods. ArmorPy's documentation is located on an external website[1]. It's also very limited and does not contain any use cases.

Table 1 shows documentation quality evaluation based on authors' experience installing, configuring and experimenting described in this article.

Table 1. Tools' documentation quality. '-' signifies a low-quality documentation, '- -' signifies a very low-quality documentation or its absence.

Tool	Documentation quality
KnowRob	-
ARMOR	- -

5 Used technologies and configuration

5.1 KnowRob

KnowRob system installation was carried out according to the instructions in the github repository[4]. As required, ROS, SWI Prolog, Mongo DB server and Rosprolog were installed first, however the Rosprolog package turned out to be problematic. Absence of the SWI-Prolog package was being detected during the catkin_make command execution, which was causing errors. It was an unexpected problem, because the missing package was previously installed. The solution to the problem turned out to be the removal of the workspace folder's contents and reinstalling them the same way.

ROS ROS's melodic version is required, which forces Ubuntu users to use 18.04 version of the OS. Due to two new released stable system versions, it is very unlikely that 18.04 will be the primary system on a user's computer. For this reason the user is forced to use a shared system or a virtual machine, which causes the loss of performance and ongoing support for the OS.

In the description of the installation process on the official ROS website[6], both required and additional steps are included, although they lack clear distinction. For a person installing this tool for the first time it may result in mistakes during the process, which might result in a situation where a full re-installation is necessary.

5.2 ARMOR

The installation of the ARMOR framework was conducted according to the information published on author's website [1]. As described, ROS (kinetic version), RosJava (kinetic version), AMOR and extension package to AMOR: AMOR services, were installed. Additionally, ARMOR message interface and ARMOR Python API were installed. Unfortunately ARMOR has not been updated since 2019, which resulted in the need of usage of a system on which the framework still works - Ubuntu 16.04 Xenial Xerus.

ROS ROS is not officially listed as required for ARMOR's installation, however its first requirement is the RosJava package, which cannot be installed without it. Since ARMOR framework has not been updated, kinetic version of ROS is necessary. The user is being led through the installation process efficiently and

instructions do not contain glaring inaccuracies. However, the description, same as with the melodic version, does not distinct between required and additional steps clearly enough.

RosJava RosJava is a ROS implementation in Java. It is a required component in the ARMOR installation process. The instructions are mostly correct, but the user is mislead in the following part while executing the command:

```
rosdep install --from-paths src -i -y
```

RosJava authors suggest to ignore warnings about some packages' absence, but use of the ARMOR framework is not possible without them, making manual installation of listed packages necessary:

```
ros-kinetic-move-base-msgs
ros-kinetic-world-canvas-msgs
ros-kinetic-scheduler-msgs
ros-kinetic-rocon-tutorial-msgs
ros-kinetic-rocon-interaction-msgs
ros-kinetic-rocon-device-msgs
ros-kinetic-rocon-app-manager-msgs
ros-kinetic-gateway-msgs
ros-kinetic-concert-service_msgs
ros-kinetic-concert-msgs
ros-kinetic-yocs-msgs
ros-kinetic-ar-track-alvar-msgs
```

ARMOR Armor is an AMOR framework extension, therefore installation of both environments is required. Many problems are caused by the general lack of installation instructions. It is not mentioned which github repositories are required to launch the environment and which are not. This resulted in downloading all packages listed in the instruction. Instruction was placed on github repository [2] describing ARMOR functioning examples. After downloading all packages and installing by the catkin_make command, the environment is ready for use.

5.3 Installation difficulty evaluation

Table 2 aggregates installation difficulty level of tools used for this article.

5.4 Configuration

Applied knowledge base configuration Both KnowRob and ARMOR were used along with the KnowRob2 ontologies - unit.owl and knowrob.owl, which contain OWL, RDF-Schema, URDF, Qudt, Quantity, Dimension, SOMA and IOLite ontologies references.

Table 2. Tool installation difficulty level. '+' signifies moderately easy installation, '-' signifies moderately hard installation.

Tool	Installation difficulty
KnowRob: ROS	+
KnowRob: RosProlog	-
ARMOR: ROS	+
ARMOR: RosJava	-
ARMOR: ARMOR	-

KnowRob In KnowRob's case unit.owl, knowrob.owl and related ontologies are autoloaded by roslaunch server startup or on CLI mode startup using rosrun thus no additional configuration is needed.

ARMOR To launch ARMOR, the roscore service needs to be ran in the first terminal window using the command 'roscore'. Roscore enables ROS nodes communication. Afterwards, 'rosrun armor execute it.emarolab.armor.ARMORMainService' has to be ran in another terminal window for the ARMOR service to launch. In a third terminal window the ontology should be loaded, which can be done in two ways:

1. By terminal and rosservice call.

```
rosservice call /armor_interface_srv "armor_request:
  client_name: 'terminal'
  reference_name: 'ref1'
  command: 'LOAD'
  primary_command_spec: 'FILE'
  secondary_command_spec: ''
  args: ['/home/user/catkin_ws/src/Mydir/owl/test.owl',
  ↪  'http://www.IRIs.org/test'
  ]"
```

2. By a Python program using functions provided by ARMOR client interface.

```
client = ArmorClient("client", "reference")
client.utils.load_ref_from_file(path + "test.owl",
↪  "http://www.IRIs.org/test")
client.utils.mount_on_ref()
client.utils.set_log_to_terminal(True)
```

6 Experiments

The experiments focused on the creation of a simulation environment, passing an example state of the environment, and the analysis of the knowledge base through analysis of responses to queries about objects in the environment. For all these cases an example part of a typical kitchen environment was created and

knowledge about a pair of kitchen objects, a cup and a cupboard, in a form of triples was added. The knowledge structure is presented on Fig. 1.

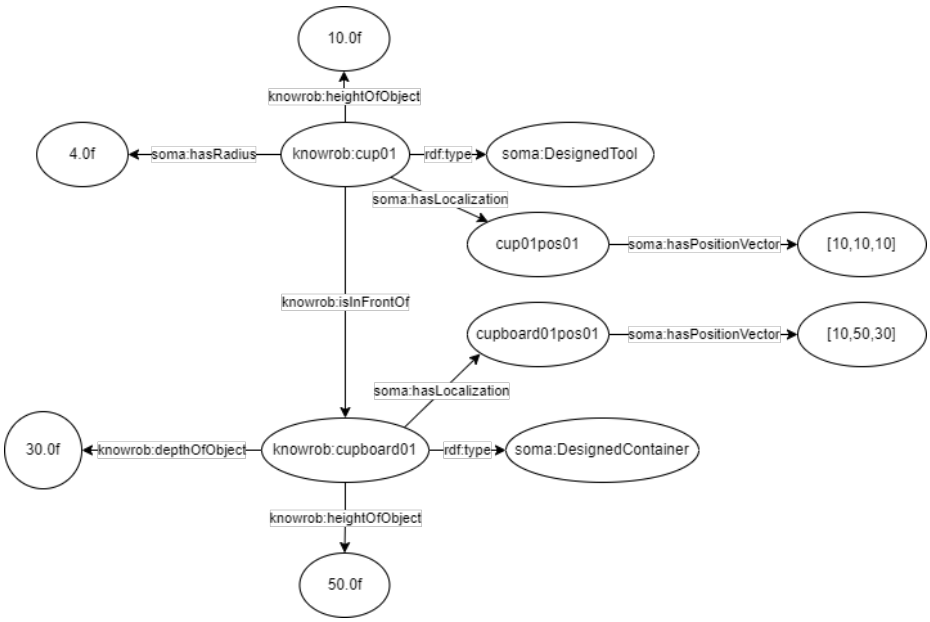

Fig. 1. Applied ontology describing a part of a typical kitchen environment.

To carry out the aforementioned experiments 3 queries were made and repeated 1000 times each in order to check the consistency of the performance. Query 1. checks the class of the cup object 'cup01'. Query 2. checks the relation between 'cup01' and the cupboard 'cupboard01'. Query 3. checks the position of the cupboard 'cupboard01' in the environment.

6.1 Knowrob

The information about the kitchen environment was written down as RosProlog queries and input into the knowledge base by execution of a python script. For this operation KnowRob's function 'rdf_db:rdf_assert()' was used. Afterwards, different RosProlog queries were ran 1000 times, also via the KnowRob framework, in order to check the state of the environment and the response times.

```
1. instance_of('cup01', A)
2. holds(knowrob:'cup01', B, knowrob:'cupboard01')
3. holds(knowrob:'cupboard01', soma:'hasLocalization', _L),
    holds(_L, soma:'hasPositionVector', Vec)
```

Table 3. The results of running 1000 queries to KnowRob (ms).

Query	Average time	Standard deviation	Variance
1	1.089	0.262	0.069
2	6.040	6.599	43.555
3	7.857	9.136	83.460

Table 3 presents the results of the queries to the KnowRob framework.

Query 1. asked only for the class of 'cup01' and took only about 1 ms to finish, while queries 2. and 3., which check the relations between objects and the location of one of them, took several times longer to complete.

6.2 ARMOR

The ARMOR framework allows for running queries in two ways:

1. Via terminal and rosservice call with SPARQL queries.

```
rosservice call /armor_interface_srv "armor_request:
  client_name: 'terminal'
  reference_name: 'ref'
  command: 'QUERY'
  primary_command_spec: 'SPARQL'
  secondary_command_spec: ''
  args: ['
PREFIX knowrob: <http://knowrob.org/kb/knowrob.owl#>
PREFIX soma: <http://www.ease-crc.org/ont/SOMA.owl#>
SELECT ?a ?b WHERE {
  ?a ?b soma:DesignedContainer
}'
]"
```

2. Via a python script using functions of the ARMOR client's interface.

```
client.query.check_ind_exists("knowrob:cup01"))
client.query.dataprop_b2_ind("knowrob:isInFrontOf",
↪  "knowrob:cup01"))
client.query.dataprop_b2_ind("soma:hasLocalization",
↪  "knowrob:cupboard01"))

print(client.query.check_ind_exists("knowrob:cup01"))
```

Queries which were ran through the terminal in the SPARQL language caused suspension of the terminal and the knowledge base. Because of this, the Python module was chosen as the method for running queries. The Python module received the data correctly, but when it came to querying it, it was behaving nondeterministically, which was caused by the suspension of execution of the

Python script. Sometimes the whole script would be executed, but most of the time the execution of the script and the knowledge base were getting suspended and only the beginning of the script was being properly executed. This behaviour made collection of the necessary comparison data impossible.

7 Conclusion

The conducted TRIZ analysis indicates what the specific needs in relation to the designed companion robot system are. Analyzed tools: KnowRob and Armor were identified by reducing the problem to a generic form and formulating generic solutions. Table 4 presents the overall evaluation of the tools' components based on what has been experienced through tests in this work.

Table 4. Tools' components' evaluation. '+ +' signifies a high-quality component, '-' signifies a low-quality component, '- -' signifies a very low-quality component.

Tool	Documentation	Installation	Query responsiveness
KnowRob	-	-	+ +
ARMOR	- -	-	- -

Due to implementation problems, lacking documentation and nondeterministic nature of the ARMOR framework, tests could not be carried out on both frameworks and compared. The presented results refer to only the KnowRob framework, which allowed for repeated querying. Queries in the ARMOR framework resulted in both returning valid responses and invalid execution errors. Due to this reason, quantitative analysis was impossible.

KnowRob carries out queries correctly. In response to them, it returns the correct class of an object, relations with another object and relational position, where querying the relation between objects takes considerably longer than a simple class checking query. In the context of carried out tests, queries are very fast, but in a real-life scenario a service robot faces a very large quantity of both simple and more complex queries, making the responsiveness a very important aspect of the framework.

In terms of documentation and available support, the KnowRob framework has a considerably bigger amount of materials available. The amount of users also contributes to the overall shared knowledge and experience, making implementation less time consuming and better tested.

Documentation of the ARMOR framework is lacking. The framework, which had the potential of being more widely used, looks to had been abandoned and is now being used mainly by users strongly determined or those who are forced to for other reasons.

References

1. Armor api documentation, http://emarolab.github.io/armor_py_api/armor_api.html
2. Armor github page, http://emarolab.github.io/armor_py_api/armor_api.html
3. Knowrob documentation, http://knowrob.org/doc
4. Knowrob github page, https://github.com/knowrob/knowrob
5. Ontology portal - sumo, http://www.ontologyportal.org/
6. Ros ubuntu installation guide, http://wiki.ros.org/melodic/Installation/Ubuntu
7. Caresses project, http://caressesrobot.org/en/
8. Antoniou, G., van Harmelen, F.: Web ontology language: Owl. Handbook on Ontologies p. 67–92 (2004). https://doi.org/10.1007/978-3-540-24750-0_4
9. Beetz, M., Beßler, D., Haidu, A., Pomarlan, M., Bozcuoğlu, A.K., Bartels, G.: Know rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 512–519 (2018). https://doi.org/10.1109/ICRA.2018.8460964
10. Burgun, A.: Desiderata for domain reference ontologies in biomedicine. Journal of Biomedical Informatics **39**(3), 307–313 (2006). https://doi.org/https://doi.org/10.1016/j.jbi.2005.09.002, https://www.sciencedirect.com/science/article/pii/S1532046405000997, biomedical Ontologies
11. Dogmus, Z., Erdem, E., Patoglu, V.: Rehabrobo-onto: Design, development and maintenance of a rehabilitation robotics ontology on the cloud. Robotics and Computer-Integrated Manufacturing **33**, 100–109 (2015). https://doi.org/https://doi.org/10.1016/j.rcim.2014.08.010, https://www.sciencedirect.com/science/article/pii/S0736584514000714, special Issue on Knowledge Driven Robotics and Manufacturing
12. Haidegger, T., Barreto, M., Gonçalves, P., Habib, M.K., Ragavan, S.K.V., Li, H., Vaccarella, A., Perrone, R., Prestes, E.: Applied ontologies and standards for service robots. Robotics and Autonomous Systems **61**(11), 1215–1223 (2013). https://doi.org/https://doi.org/10.1016/j.robot.2013.05.008, https://www.sciencedirect.com/science/article/pii/S092188901300105X, ubiquitous Robotics
13. Mann, D.: Business Matrix 3.0: Solving Management, People & Process Contradictions. IFR Press (2018)
14. Mendelson, E.: Introduction to mathematical logic (1987). https://doi.org/10.1007/978-1-4615-7288-6
15. Menzel, C.: Reference ontologies - application ontologies: Either/or or both/and?. (01 2003)
16. Olivares-Alarcos, A., Beßler, D., Khamis, A., Goncalves, P., Habib, M.K., Bermejo-Alonso, J., Barreto, M., Diab, M., Rosell, J., Quintas, J., et al.: A review and comparison of ontology-based approaches to robot autonomy. The Knowledge Engineering Review **34**, e29 (2019). https://doi.org/10.1017/S0269888919000237
17. Olszewska, J.I., Barreto, M., Bermejo-Alonso, J., Carbonera, J., Chibani, A., Fiorini, S., Goncalves, P., Habib, M., Khamis, A., Olivares, A., de Freitas, E.P., Prestes, E., Ragavan, S.V., Redfield, S., Sanz, R., Spencer, B., Li, H.: Ontology for autonomous robotics. In: 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). pp. 189–194 (2017). https://doi.org/10.1109/ROMAN.2017.8172300

18. Ruy, F., Guizzardi, G., Falbo, R., Reginato, C., Dos Santos, V.A.: From reference ontologies to ontology patterns and back. Data & Knowledge Engineering **109**, 41–69 (03 2017). https://doi.org/10.1016/j.datak.2017.03.004
19. Sikos, L.: Vidont: a core reference ontology for reasoning over video scenes. Journal of Information and Telecommunication **2**, 1–13 (02 2018). https://doi.org/10.1080/24751839.2018.1437696
20. Suh, I.H., Lim, G.H., Hwang, W., Suh, H., Choi, J.H., Park, Y.T.: Ontology-based multi-layered robot knowledge framework (omrkf) for robot intelligence. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 429–436 (2007). https://doi.org/10.1109/IROS.2007.4399082
21. Uschold, G.: Ontologies: Principles methods and applications. The Knowledge Engineering Review **11**(2) (1996)