IEEE STANDARDS ASSOCIATION

IEEE Standard Ontologies for Robotics and Automation

IEEE Robotics and Automation Society

Sponsored by the
Standing Committee for Standards Activities

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1872™-2015

IEEE Standard Ontologies for Robotics and Automation

Sponsor

Standing Committee for Standards Activities
of the
IEEE Robotics and Automation Society

Approved 16 February 2015

IEEE-SA Standards Board

Abstract: A core ontology that specifies the main, most general concepts, relations, and axioms of robotics and automation (R&A) is defined in this standard, which is intended as a reference for knowledge representation and reasoning in robots, as well as a formal reference vocabulary for communicating knowledge about R&A between robots and humans. This standard is composed of a core ontology about R&A, called CORA, together with other ontologies that give support to CORA.

Keywords: automation, core ontology, IEEE 1872™, methodology, ontology, robotics

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-7381-9650-3 STD20183
Print: ISBN 978-0-7381-9651-0 STDPD20183

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading "Important Notice" or "Important Notices and Disclaimers Concerning IEEE Standards Documents."

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association ("IEEE-SA") Standards Board. IEEE ("the Institute") develops its standards through a consensus development process, approved by the American National Standards Institute ("ANSI"), which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied "AS IS" and "WITH ALL FAULTS."

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

Photocopies

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at http://ieeexplore.ieee.org/xpl/standards.jsp or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at http://standards.ieee.org.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: http://standards.ieee.org/findstds/errata/index.html. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this IEEE standard was completed, the Ontologies for Robotics and Automation (ORA) Working Group had the following membership:

Craig Schlenoff, *Chair*
Edson Prestes, *Vice Chair*
Paulo Jorge Sequeira Gonçalves, *Secretary*

Mara Abel
Yacine Amirat
Stephen Balakirsky
Marcos Ennes Barreto
Joel Luis Carbonera
Abdelghani Chibani
Sandro Rama Fiorini
Sébastien Gérard
Vitor Augusto Machado Jorge
Maki Habib
Tamás Haidegger
Sampath Kumar
Howard Li
Angela Locoro
Raj Madhavan
Veera Ragavan
Signe Redfield
Vitor Fortes Rey

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Francesco Amigoni
Stephen Balakirsky
Marcos Ennes Barreto
Joel Luis Carbonera
Abdelghani Chibani
Keith Chow
Sandro Rama Fiorini
Paulo Goncalves
Randall Groves
Tamás Haidegger
Werner Hoelzl
Noriyuki Ikeuchi
Vitor Augusto Machado Jorge
Piotr Karocki
Thomas Kramer
Howard Li
Raj Madhavan
Charles Ngethe
Signe Redfield
Robert Robinson
Veera Ragavan
Craig Schlenoff
Edson Silva, Jr.
Walter Struppler
Marcy Stutzman
David Tepen
Daidi Zhong

When the IEEE-SA Standards Board approved this standard on 16 February 2015, it had the following membership:

John Kulick, *Chair*
Jon Walter Rosdahl, *Vice Chair*
Richard H. Hulett, *Past Chair*
Konstantinos Karachalios, *Secretary*

Peter Balma
Farooq Bari
Ted Burse
Clint Chaplin
Stephen Dukes
Jean-Philippe Faure
Gary Hoffman
Michael Janezic
Jeffrey Katz
Joseph L. Koepfinger*
David J. Law
Hung Ling
Oleg Logvinov
T. W. Olsen
Glenn Parsons
Ron Petersen
Adrian Stephens
Peter Sutherland
Yatin Trivedi
Phil Winston
Don Wright
Yu Yuan

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Patrick Gibbons
IEEE-SA Content Production and Management

Michael Kipness
IEEE-SA Technical Program Operations

Introduction

This introduction is not part of IEEE Std 1872-2015, IEEE Standard Ontologies for Robotics and Automation.

Seamless and unambiguous communication between people demands a common, well-defined vocabulary. Otherwise, misinterpretations can happen and no information or—even worse—incorrect information can be exchanged between the participants, often with negative consequences. This could happen when two people who do not speak the same language try to communicate. The same applies to human-robot and robot-robot communication, where an intermediate standard language with clear and well-defined terms is a sine qua non condition for common understanding.

The growing complexity of behaviors that robots are expected to present naturally entails the use of increasingly complex knowledge as well as the need for multi-robot and human-robot collaboration. In this context, the need for a standard and well-defined model for capturing this knowledge is becoming evident. The existence of such a standard knowledge model, precisely defining the concepts of the robotics domain, will help ensure common understanding among various stakeholders involved in the lifecycle of robotic systems, enabling efficient and reliable data integration and information exchange among them.

Ontology plays a fundamental role in this context. It formally specifies the key concepts, properties, relationships, and axioms of a given domain. Unlike taxonomies, which provide only a set of vocabulary and a single type of relationship between terms, an ontology provides a richer set of relationships, constraints, and rules. In general, ontologies make the relevant knowledge about a domain explicit in a computer-interpretable format, allowing software to reason over that knowledge to infer new information. Furthermore, ontologies are a great tool for diminishing the ambiguity in knowledge transfer among groups of humans, robots, and other artificial systems that share the same conceptualization.

In this sense, the Ontologies for Robotics and Automation Working Group (ORA WG) is actively working with industry, academia, and government organizations to develop a set of ontologies and an associated modeling methodology to be used as a standard in robotics and automation (R&A). As it is extremely difficult to develop a single ontology that covers the entire scope of R&A, the ORA WG decided to focus initially on two subdomains: industrial robotics and service robotics. This decision was based on the prevalence of robots in these markets and the standardization necessities accompanying them. Therefore, ORA WG comprises three subgroups, two of them associated with each of the above subdomains. The third subgroup, called Upper Ontology/Methodology (UpOM), is in charge of the development of a more general ontology to bring all of the subdomain ontologies together. This document is the result of the work done by the UpOM and presents a core ontology for R&A (CORA), which specifies the general notions underlying R&A and aims to provide clear definitions of the common concepts that will permeate all derived ontologies to be developed within the ORA WG. Thus, CORA focuses on defining what a robot is in the scope of the standard, along with the specification of other related entities.

This document also includes other ontologies complementing CORA. The ontology CORAX represents concepts and relations commonly found in R&A subdomains but that are too general to be included in CORA. The ontology RPARTS includes concepts useful to represent robot parts. Finally, the ontology POS captures general notions about position and orientation.

Contents

IEEE Standard Ontologies for Robotics and Automation

1. Overview

1.1 Scope

This standard defines a core ontology that allows for the representation of, reasoning about, and communication of knowledge in the robotics and automation (R&A) domain. This ontology includes generic concepts as well as their definitions, attributes, constraints, and relationships. These terms can be specialized to capture the detailed semantics for concepts in robotics sub-domains.

This standard contains the Core Ontology for Robotics and Automation (CORA) with the representation of fundamental concepts from which the more detailed concepts belonging to other Ontologies for Robotics and Automation Working Group (ORA WG) ontologies are constructed. This standard also defines the ontology engineering methodology used to construct the ORA ontologies.

1.2 Purpose

The purpose of this standard is to provide a methodology for knowledge representation and reasoning in robotics and automation (R&A) together with the core ontology for the R&A domain. The standard provides a unified way of representing knowledge and provides a common set of term definitions, allowing for unambiguous knowledge transfer among any group of humans, robots, and other artificial systems.

The standard aims to provide a common vocabulary along with clear and concise definitions from the R&A domain. With the growing complexity of behaviors that robots are expected to perform, as well as the need for multi-robot collaboration and human-robot collaboration, the need for a standard and well-defined

knowledge representation is becoming more evident. The standard knowledge representation methodology and terminology:

a) More precisely define the concepts in the robot's knowledge representation

b) Promote common understanding among members of the community

c) Facilitate data integration and transfer of information among robotic systems

Information included in this knowledge representation encompasses, but is not limited to, robot hardware and software, activities and goals, environment, cause and effects of performing actions, and relationship among other robots and people.

The intended audience for this standard is robot manufacturers, system integrators, robot end users (part manufacturers, automotive industry, construction industry, service and solution providers, etc.), robot equipment suppliers, robot software developers, and researchers/developers.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

Huang, H.-M, E. Messina, and J. Albus, "Toward a Generic Model for Autonomy Levels for Unmanned Systems (ALFUS)," Performance Metrics for Intelligent Systems (PerMIS) Workshop, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2003.[1]

ISO 8373:2012, Robots and robotic devices—Vocabulary.[2]

Niles, I. and A. Pease, "Towards a standard upper ontology," FIOS '01: Proceedings of the International Conference on Formal Ontology in Information Systems, pp. 2–9, 2001.[3]

Pease, A., "Standard upper ontology knowledge interchange format," Unpublished language manual, 2004.[4]

Suggested Upper Merged Ontology (SUMO), Accessed August 2014.[5]

[1] NIST publications are available from the National Institute of Standards and Technology, 100 Bureau Drive, Stop 1070, Gaithersburg, MD 20899-2300, USA (http://www.nist.gov/index.html).
[2] ISO publications are available from the ISO Central Secretariat, 1, ch. de la Voie-Creuse, CP 56, CH-1211 Geneva 20, Switzerland (http://www.iso.org/). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).
[3] Available at the ACM Digital Library at: http://dl.acm.org/citation.cfm?id=505170.
[4] Available at http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/sigma/suo-kif.pdf.
[5] Available at http://www.ontologyportal.org.

3. Definitions, abbreviations, and acronyms

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.[6]

Some definitions refer to concepts and relations from the SUMO ontology.[7]

artificial system: An artifact (*Artifact* in SUMO) formed by various interacting devices (*Device* in SUMO) and other objects (*Object* in SUMO) in order to execute a function.

automated robot: A role for a robot performing a given task in which the robot acts as an automaton, not adapting to changes in the environment and/or following scripted plans. *Contrast:* **fully autonomous robot**; **remote-controlled robot**; **semi-autonomous robot**; **teleoperated robot**.

collective robotic system: A robotic system having a robot group as part. *See also:* **robot group**; **robotic system**. *Contrast:* **single robotic system**.

coordinate system: An abstract entity (*Abstract* in SUMO) used for specifying location and orientation that is defined in relation to a single reference object (*Object* in SUMO). Coordinate systems are related through hierarchies (i.e., trees). For instance, the local coordinate system of a robot is referenced by the robot itself. The reference object is not necessarily the origin of the coordinate system. A coordinate system defines at least one dimension in which points get their coordinate values. Points in a given coordinate system can be mapped to other coordinate systems by means of a transformation. *See also:* **global coordinate system**; **local coordinate system**; **transformation**.

design: A proposition (*Proposition* in SUMO) that abstracts the structure of one or more artifacts (*Artifact* in SUMO). A design is used to abstract information in contexts such as industrial robotics. A design is different from a blueprint; a blueprint represents a particular design.

fully autonomous robot: A role for a robot performing a given task in which the robot solves the task without human intervention while adapting to operational and environmental conditions. *Contrast:* **automated robot**; **remote-controlled robot**; **semi-autonomous robot**, **teleoperated robot**.

global coordinate system: An arbitrary coordinate system chosen by an agent as the global reference frame that constitutes the global coordinate system for that agent. In a hierarchy of local coordinate systems, the global coordinate system is the root of a tree of local coordinate systems. *See also:* **coordinate system**. *Contrast:* **local coordinate system**.

interaction: A process (*Process* in SUMO) in which two agents participate (*Agent* in SUMO). It is composed by two subprocesses defining action and reaction. The action subprocess initiated by agent X on a patient agent Y causes a reaction subprocess having Y as agent and X as patient.

local coordinate system: A coordinate system bounded to a hierarchical structure of coordinate systems and that is not at the root of the structure. *See also:* **coordinate system**. *Contrast:* **global coordinate system**.

[6] *IEEE Standards Dictionary Online* subscription is available at: http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.
[7] Information on references can be found in Clause 2.

orientation measure: Essentially a measure (*Measure* in SUMO) attributed to a (physical) object (*Object* in SUMO) concerning information regarding where the object is pointing to in relation to the reference object of the orientation coordinate system. *See also:* **coordinate system**; **orientation value**; **orientation region**; **pose**; **robot**.

orientation region: Defines a region or interval orientation in relation to a reference object (*Object* in SUMO). For instance, the "south" interval of a compass constitutes an orientation region in the one-dimensional, circular coordinate system of the compass. Eventually, position regions and orientation regions are referred by similar words. For instance, it is valid to say that a robot is at the north position, facing north. The former relates to a position region, i.e., the north region of a given country; the later relates to an orientation region, i.e., the orientation interval around north on the compass. *See also:* **orientation measure**; **position region**. *Contrast:* **orientation point**.

orientation value: A value in a coordinate system denoting a specific orientation. Orientation values in one coordinate system can be mapped to other coordinate systems. An example of use of orientation value is in "the robot is oriented 54° in relation to the reference object." *See also:* **orientation measure**. *Contrast:* **orientation region**.

physical environment: A physical environment is an object (*Object*, in SUMO) that has at least one specific part: a region (*Region* in SUMO) in which it is located. In addition, a physical environment relates to at least one reference object (*Object* in SUMO) based on which region is defined.

pose: A position and an orientation constitute a pose. The pose of an object is the description of any position and orientation measurements of that object. *See also:* **orientation measure**; **position measure**.

position measure: A measure (*Measure* in SUMO) attributed to a (physical) object (*Object* in SUMO) describing its position. A position can be described by a point or a region. For instance, one can describe a robot as positioned at coordinates (x, y) in the coordinate system, or at the front of the box, where "front" comprises a conical region centered on the box and pointed forward. *See also:* **coordinate system**; **pose**; **position point**; **position region**; **robot**.

position point: A point in a position coordinate system. It denotes a precise indication of position of a given object. Position points are always defined in a single coordinate system. *See also:* **position measure**. *Contrast:* **position region**.

position region: An abstract region in a position coordinate system. More specifically, a position region is defined by position points in a given coordinate system. It defines qualitative positions such as "left of," "in front of," "on top of," etc. These expressions define regions in relation to a reference object in which other objects are placed. A position region is always generated by a given spatial operator applied to a list of reference objects. *See also:* **position measure**. *Contrast:* **position point**.

processing device: An electric device (*Electric Device* in SUMO) whose purpose is to serve as an instrument in a subclass of computer process (*Computer Process* in SUMO).

remote-controlled robot: A role for a robot performing a given task in which the human operator controls the robot on a continuous basis, from a location off the robot, via only her/his direct observation. In this mode, the robot takes no initiative and relies on continuous or nearly continuous input from the human operator. *Contrast:* **automated robot**; **fully autonomous robot**; **semi-autonomous robot**; **teleoperated robot**.

robot actuating part: A role for devices (*Device* in SUMO) that allow for the robot to move and act in the surrounding environment. *Contrast:* **robot communicating part**; **robot processing part**; **robot sensing part**. *See also:* **robot part**.

robot communicating part: A role for devices (*Device* in SUMO) that serves as instruments in a robot-robot communication process or a human-robot communication process by allowing the robot to send (or receive) information to (or from) a robot or a human. *Contrast:* **robot actuating part**; **robot processing part**; **robot sensing part**. *See also:* **robot part**.

robot group: A group (*Group* in SUMO) of robots organized to achieve at least one common goal. *See also:* **robot**.

robot interface: A device (*Device* in SUMO) composed by the devices that play the roles of sensing parts, actuating parts, and communicating parts. Through the interface, the robot can sense and act on the environment as well as communicate with other agents. Therefore, the robot interface can be viewed as way to refer to all the devices that allow the robot to interact with the world. Each robot has one and only one robot interface. *See also:* **robot actuating part**; **robot communicating part**; **robot sensing part**.

robot part: A role played by any device (*Device* in SUMO) that is attached to the robot and serves in the functioning of the robot. Devices that are considered robot parts while attached to a robot are not necessarily always a robot part in an ontological sense, since they exist by themselves and, in most cases, they can be connected to other kinds of devices. For instance, a power source is essentially a device; however, a specific instance of a power source can be dynamically considered as a robot part during a specific time interval while connected to a robot. The parts of the devices that are considered robot parts are considered robot parts as well. *See also:* **robot**; **robot actuating part**; **robot communicating part**; **robot processing part**; **robot sensing part**.

robot processing part: A role played by processing devices which allows the robot to process information. *Contrast:* **robot actuating part**; **robot communicating part**; **robot sensing part**. *See also:* **processing device**; **robot part**.

robot sensing part: A role played by any measuring device (*MeasuringDevice* in SUMO) that allows the robot to acquire information about its environment. *Contrast:* **robot actuating part**; **robot communicating part**; **robot processing part**. *See also:* **robot part**.

robot: An agentive device (*Agent* and *Device* in SUMO) in a broad sense, purposed to act in the physical world in order to accomplish one or more tasks. In some cases, the actions of a robot might be subordinated to actions of other agents (*Agent* in SUMO), such as software agents (bots) or humans. A robot is composed of suitable mechanical and electronic parts. Robots might form social groups, where they interact to achieve a common goal. A robot (or a group of robots) can form robotic systems together with special environments geared to facilitate their work. *See also:* **automated robot**; **fully autonomous robot**; **remote-controlled robot**; **robot group**; **robotic system**; **semi-autonomous robot**; **teleoperated robot**.

robotic environment: A physical environment equipped with a robotic system. *See also:* **physical environment**; **robotic system**.

robotic system: An artificial system formed by one or more robots (single robots or groups of robots) and at least one device (*Device* in SUMO) supporting the operation of the robot(s). *See also:* **artificial system**; **collective robotic system**; **robot**; **single robotic system**.

semi-autonomous robot: A role for a robot performing a given task in which the robot and a human operator plan and conduct the task, requiring various levels of human interaction. *Contrast:* **automated robot**; **fully autonomous robot**; **remote-controlled robot**; **teleoperated robot**.

single robotic system: A robotic system having one and only one robot as part and one or more supporting devices. *See also:* **robotic system**. *Contrast:* **collective robotic system**.

spatial operator: A mathematical function that can map reference objects (*Object* in SUMO) to regions in a coordinate system. *See also:* **coordinate system**.

teleoperated robot: A role for a robot performing a given task in which a human operator, using sensory feedback, either directly controls the actuators or assigns incremental goals on a continuous basis, from a location off the robot. A teleoperated robot will complete its last command after the operator stops sending commands, even if that command is complex or time-consuming. *Contrast:* **automated robot**; **fully autonomous robot**; **remote-controlled robot**; **semi-autonomous robot**.

transformation: A function that maps values in a coordinate system to another coordinate system. *See also:* **coordinate system**.

3.2 Abbreviations and acronyms

ALFUS	Autonomy Levels for Unmanned Systems
CORA	Core Ontology for Robotics and Automation
ORA	Ontology for Robotics and Automation
ORA WG	ORA Working Group
POS	The ontology that captures general notions about position and orientation (not an acronym per se)
R&A	Robotics and Automation
SUMO	Suggested Upper Merged Ontology

4. Conventions

The reminder of this document presents the formal definitions of CORA and additional ontologies. The formal definitions are written as SUO-KIF formulas (see Pease, "Standard upper ontology knowledge interchange format" [8]), such that:

a) Underlined terms refer to entities of the ontologies defined in this standard.

b) Non-underlined terms refer to entities of SUMO or primitives of SUO-KIF.

5. SUMO

SUMO ontology is a top-level ontology that defines basic ontological categories across all domains. This section gives only a brief overview of its main concepts, illustrated in Figure 1. Check the SUMO Ontology portal and Niles and Peace, "Towards a standard upper ontology, 2001, for more information.

[8] Information on references can be found in Clause 2.

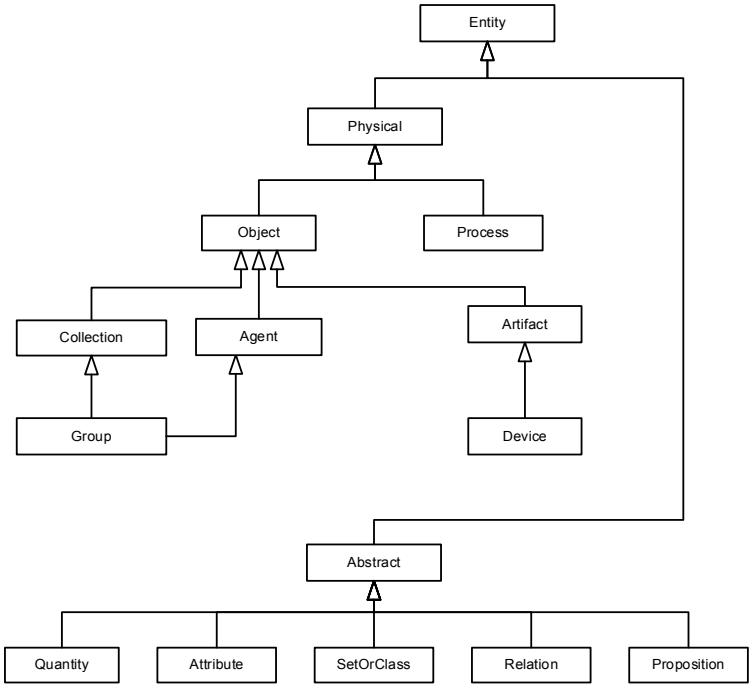

Figure 1— Basic SUMO taxonomy.

The main SUMO category is *Entity*, which is a disjoint partition of *Physical* and *Abstract* concepts. *Physical* represents entities that have spatio-temporal extension. *Abstract* describes entities that do not have or are chosen not to have spatio-temporal extension.

Physical is further partitioned into *Object* and *Process*. *Object* exists in space, having spatial parts parallel in time. *Process* is the class of individual entities that happen in time and have temporal parts or stages. That means SUMO follows an endurantist perspective instead of a perdurantist one. For an endurantist, an object keeps its identity through time and so, while some processes might change things about it, every part that is essential to it is always present. On the other hand, for a perdurantist, an object is composed of every temporal part it has at all times, so all things about it are indexed in time. A good analogy is to think that perdurantists see things as regions in a 4D space while endurantists see them as 3D things that can change in processes.

Abstract is further partitioned into *Quantity*, *Attribute*, *SetOrClass*, *Relation* and *Proposition*. *Quantity* abstracts numeric and physical quantities. *Attribute* abstracts qualities that cannot or are chosen not to be reified as subclasses of *Object*. *SetOrClass* abstracts entities that have elements or instances. *Relation* abstracts *n*-ary relations, functions and lists. Finally, *Proposition* expresses a complete thought or a set of such thoughts, which can be represented by entities such as a formula or a drawing.

Below the reader will find the definition of terms, according to the SUMO version 1.52, which will be used in the definition of CORA terms.

— *Abstract:* Properties or qualities as distinguished from any particular embodiment of the properties/qualities in a physical medium. Instances of *Abstract* can be said to exist in the same sense as mathematical objects such as sets and relations, but they cannot exist at a particular place and time without some physical encoding or embodiment.

— *Agent*: Something or someone that can act on its own and produce changes in the world.

— *Artifact*: An object that is the product of a making.

— *Attribute:* Qualities that we cannot or choose not to reify into subclasses of object.

— *Binary Predicate:* A predicate relating two items—its valence is two.

— *Binary Relation:* Binary relations are relations that are true only for pairs of things. Binary relations are represented as slots in frame systems.

— *Collection:* Collections have members like classes, but, unlike classes, they have a position in space-time and members can be added and subtracted without thereby changing the identity of the collection. Some examples are toolkits, football teams, and flocks of sheep.

— *Computer Process:* An instance of computer process is a process which manipulates data in the computer.

— *Device:* A device is an artifact whose purpose is to serve as an instrument in a specific subclass of a process.

— *Electric Device:* A device that uses electricity as its primary power source.

— *Entity:* The universal class of individuals. This is the root node of the ontology.

— *Function:* A function is a term-forming relation that maps from an *n*-tuple of arguments to a range and that associates this *n*-tuple with at most one range element. Note that the range is a set or class, and each element of the range is an instance of the set or class.

— *Group:* A collection of agents, e.g., a flock of sheep, a herd of goats, or the local Boy Scout troop.

— *Measure:* A very general predicate for asserting that a particular object is measured by a particular physical quantity. In general, the second argument of this predicate will be a term produced with the function *Measure*.

— *Measuring Device:* Any device whose purpose is to measure a physical quantity.

— *Mechanical Joint:* A device that links two parts of a physical system and allows them to move in relation to one another. Examples include hinges, drawer slides, and ball joints.

— *Object:* Corresponds roughly to the class of ordinary objects. Examples include normal physical objects, geographical regions, locations of processes, and the complement of objects in the physical class.

— *Physical:* An entity that has a location in space-time. Note that locations are themselves understood to have a location in space-time.

— *Physical Quantity:* A physical quantity is a measure of some quantifiable aspect of the modeled world, such as “the earth’s diameter” (a constant length) and “the stress in a loaded deformable solid” (a measure of stress, which is a function of three spatial coordinates).

— *Power Source:* A source of electrical power.

— *Process:* The class of things that happen and have temporal parts or stages. Examples include extended events like a football match or a race, actions like pursuing and reading, and biological processes. The formal definition is: anything that occurs in time but is not an object.

— *Property:* This predicate holds between an instance of *Entity* and an instance of *Attribute*.

— *Proposition:* Propositions are abstract entities that express a complete thought or a set of such thoughts. As an example, the formula "(instance Yojo Cat)" expresses the proposition that the entity named Yojo is an element of the class Cat.

— *Quantity:* Any specification of how many or how much of something there is. Accordingly, there are two subclasses of quantity: number (how many) and physical quantity (how much).

— *Region:* A topographic location. Regions encompass surfaces of objects, imaginary places, and geographic areas. Note that a region is the only kind of object that can be located at itself.

— *Relation:* The class of relations. There are three kinds of relation: predicate, function, and list. Predicates and functions both denote sets of ordered *n*-tuples. The difference between these two classes is that predicates cover formula-forming operators, while functions cover term-forming operators. A list, on the other hand, is a particular ordered *n*-tuple.

— *Set* or *Class:* The set or class of sets and classes; i.e., any instance of abstract that has elements or instances.

— *Single-Valued Relation:* A relation is a single-valued relation just in case an assignment of values to every argument position except the last one determines at most one assignment for the last argument position.

— *Ternary Predicate:* The class of predicates that require exactly three arguments.

— *Time Interval:* An interval of time. Note that a time interval has both an extent and a location on the universal timeline.

— *Time Measure:* The class of temporal durations and positions of time points and time intervals along the universal timeline.

— *Time Point:* An extensionless point on the universal timeline. The time points at which processes occur can be known with various degrees of precision and approximation, but conceptually time points are point-like and not interval-like.

— *Unary Function:* The class of functions that require a single argument.

— *Universal Joint:* A joint that couples two shafts at variables angles to one another. It is distinct from a constant-velocity joint in that the shafts do not travel at a constant velocity with respect to one another. This causes vibration and wear. However, universal joints are simpler to make than constant-velocity joints.

— *Wheel:* A circular artifact which is a component of land vehicles and of some devices.

6. CORAX axioms

6.1 Background

The CORAX ontology defines concepts that are too general to be in the CORA ontology, effectively extending the coverage of the set ontologies that compose this standard. These concepts cover aspects of reality that are necessary for modelling, but are not explicitly or completely covered by SUMO.

6.2 Design

The notion of product *Design* is important across many domains. It is used to abstract information in contexts, such as industrial robotics.

A design is a *Proposition* (see Figure 2):

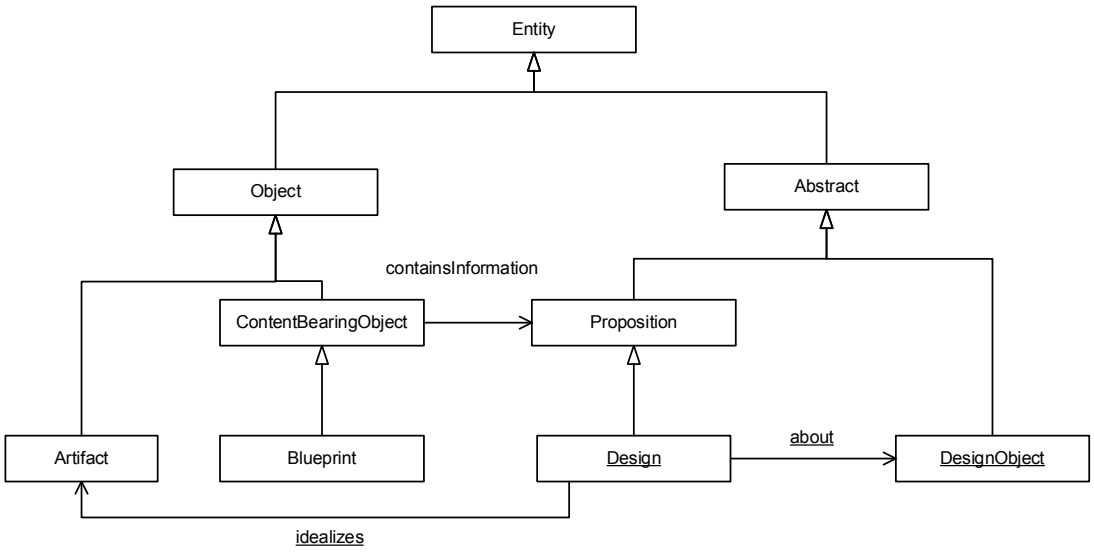

Figure 2—Concepts related to Design and DesignObject.

```
(subclass Design Proposition)
```

A design idealizes the intended structure of one or more artifacts.

```
(instance idealizes BinaryPredicate)
(instance idealizes AsymmetricRelation)
(domain idealizes 1 Design)
(domain idealizes 2 Artifact)
```

Designs are *about* design objects. Design objects are abstract idealizations of the individual artifacts conforming to the design. The relation *about* asserts that a proposition states facts about some entity. In CORAX, this relation represents that a design states facts about a design object.

```
(subclass DesignObject Abstract)

(instance about BinaryRelation)
(instance about AsymmetricRelation)
(domain about 1 Proposition)
(domain about 2 Entity)

(=>
    (instance ?O DesignObject)
    (exists (?D)
        (and
            (instance ?D Design)
            (about ?D ?O))))

(=>
    (instance ?D Design)
    (exists (?O)
        (and
            (instance ?O DesignObject)
            (about ?D ?O))))
```

Design objects bear attributes and measurements that are expected to be found in the produced artifacts. Abstract counterparts of the relations *Attribute* and *Measure* capture these notions.

```
(subrelation designAttribute property)
(instance designAttribute BinaryRelation)
(instance designAttribute AsymmetricRelation)
(domain designAttribute 1 DesignObject)
(domain designAttribute 2 Attribute)


(instance designMeasure BinaryRelation)
(instance designMeasure AsymmetricRelation)
(domain designMeasure 1 DesignObject)
(domain designMeasure 2 PhysicalQuantity)
```

Design objects may have other design objects as parts, e.g., reflecting the structure of the designed artifact.

```
(=>
    (and
        (instance ?O DesignObject)
        (abstractPart ?P ?O))
    (instance ?P DesignObject))
```

SUMO states propositions can be represented by *content bearing physicals*, such as a linguistic sentence (including logical sentences), a document, a diagram, or a computer CAD file. This standard does not put any restriction on the kinds of content bearing physicals that can represent a design.

6.3 PhysicalEnvironment

A *PhysicalEnvironment* is an object that has at least one specific part: a region in which it is located. In addition, a physical environment relates to at least one reference object based on which its region is defined (Figure 3).

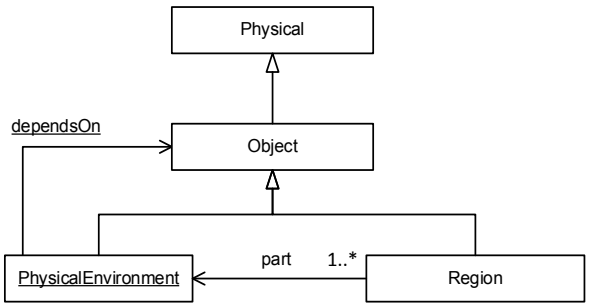

Figure 3—Concepts related to PhysicalEnvironment.

```
(subclass PhysicalEnvironment Object)

(instance dependsOn BinaryRelation)
(domain dependsOn 1 Entity)
(domain dependsOn 2 Entity)

(=>
    (instance ?ENV PhysicalEnvironment)
    (exists (?R ?O)
        (and
            (instance ?R Region)
            (part ?R ?ENV)
            (instance ?O Object)
            (dependsOn ?ENV ?O))))
```

For instance, the environment of an empty room has the actual room as a reference object and includes the minimal region in which the room is fully located. In another example, an underwater robot operates in an environment bounded by the sea floor and the sea surface in the vertical dimension, but only by abstract measurements in the horizontal plane. In this case, the reference objects that define the environment are the sea floor, the sea surface, and the landmarks used to define the horizontal plane boundary within which the vehicle will operate.

It is possible to refer to given sub-regions of maximal regions that are parts of the environment. For example, consider a given indoor environment of a house. The rooms of this house are located in their specific regions, which are parts of the maximal region which fully encompasses the house—which is also part of this environment.

All objects that are part of a physical environment are located in a region that is part of that environment.

```
(=>
    (instance ?ENV PhysicalEnvironment)
    (exists (?R)
        (and
            (instance ?R Region)
            (part ?R ?ENV)
            (forall (?O)
                (=>
                    (part ?O ?ENV)
                    (located ?O ?R))))))
```

6.4 Interaction

SUMO does not define the notion of *Interaction*. An interaction is a *Process* (Figure 4):

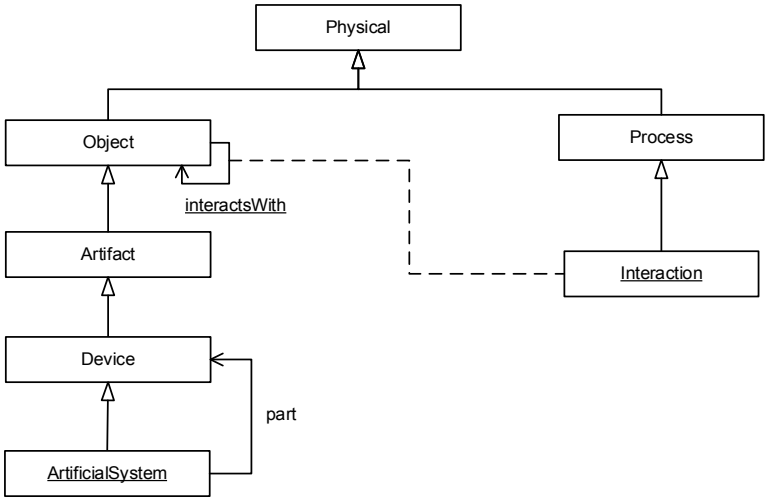

Figure 4—Concepts related to ArtificialSystem and Interaction.

```
(subclass Interaction Process)
```

It is a process in which *two* agents participate. It is composed by two subprocesses defining action and reaction. The action subprocess initiated by agent x on a patient agent y causes a reaction subprocess having y as agent and x as patient.

```
(=>
    (instance ?INT  Interaction)
    (exists (?O1 ?O2 ?P1 ?P2)
        (and
            (instance ?P1 Process)
            (instance ?P2 Process)
            (subProcess ?P1 ?INT)
            (subProcess ?P2 ?INT)
            (instance ?O1 Object)
            (instance ?O2 Object)
```

```
            (agent ?O1 ?P1)
            (patient ?O2 ?P1)
            (agent ?O2 ?P2)
            (patient ?O1 ?P2)
            (causes ?P1 ?P2))))
```

If there is at least one interaction process in which two objects participate as agents, there is an *interaction* relationship between them:

```
(instance interactsWith BinaryRelation)
(instance interactsWith SymmetricRelation)
(instance interactsWith IntransitiveRelation)
(domain interactsWith 1 Object)
(domain interactsWith 2 Object)

(<=>
    (and
        (instance ?INT  Interaction)
        (instance ?C1 CaseRole)
        (instance ?C2 CaseRole)
        (playsRoleInEvent ?O1 ?C1 ?INT)
        (playsRoleInEvent ?O2 ?C2 ?INT))
    (interactsWith ?O1 ?O2))
```

6.5 ArtificialSystem

An artificial system is an *Artifact* (Figure 4) formed by various *devices* (and other objects) that *interact* in order to execute a function.

```
(subclass ArtificialSystem Artifact)
```

For any part of an artificial system, there is at least one other part it interacts with.

```
(=>
    (instance ?S ArtificialSystem)
    (forall (?P1)
        (=>
            (part ?P1 ?S)
            (exists (?P2)
               (and
                   (part ?P2 ?S)
                   (interactsWith ?P1 ?P2)
                   (not (equal ?P1 ?P2)))))))
```

6.6 ProcessingDevice

A *ProcessingDevice* is an *ElectricDevice* whose purpose is to serve as an *instrument* in a subclass of computer process.

```
(subclass ProcessingDevice ElectricDevice)

(=>
    (instance ?DEVICE ProcessingDevice)
    (exists (?PROC)
        (and
```

```
            (subclass ?PROC ComputerProcess)
            (hasPurpose ?DEVICE
                (exists (?INST)
                    (and
                        (instance ?INST ?PROC)
                        (instrument ?INST ?DEVICE)))))))
```

6.7 RobotMotion

An occurrence of *RobotMotion* is any process of movement where the agent is a robot and the patient is one of its (robot) parts. Given that, it is any process in which the robot moves one of its parts.

```
(subclass RobotMotion Motion)

(=>
    (instance ?MOTION RobotMotion)
    (exists (?AGENT ?PATIENT)
        (and
            (instance ?AGENT Robot)
            (agent ?MOTION ?AGENT)
            (instance ?PATIENT Device)
            (patient ?MOTION ?PATIENT)
            (moves ?MOTION ?PATIENT)
            (robotPart ?PATIENT ?AGENT))))
```

6.8 HumanRobotCommunication

An occurrence of *HumanRoboCommunication* is any process that involves the transfer of information between humans and robots.

```
(subclass HumanRobotCommunication ContentBearingProcess)

(=>
    (instance ?COMMUNICATION HumanRobotCommunication)
    (exists (?ROBOT ?HUMAN)
        (and
            (instance ?ROBOT Robot)
            (instance ?HUMAN Human)
            (or
                (and
                    (agent ?COMMUNICATION ?ROBOT)
                    (destination ?COMMUNICATION ?HUMAN))
                (and
                    (agent ?COMMUNICATION ?HUMAN)
                    (destination ?COMMUNICATION ?ROBOT))))))
```

6.9 RobotRobotCommunication

An occurrence of *RobotRobotCommunication* is any process that involves the information transfer between two or more robots.

```
(subclass RobotRobotCommunication ContentBearingProcess )


(=>
    (instance ?COMMUNICATION RobotRobotCommunication)
    (exists (?ROBOT1 ?ROBOT2)
        (and
            (instance ?ROBOT1 Robot)
            (instance ?ROBOT2 Robot)
            (agent ?COMMUNICATION ?ROBOT1)
            (destination ?COMMUNICATION ?ROBOT2))))
```

7. CORA Axioms

7.1 Robot

The main concept in CORA is *Robot*. It is related to most of the concepts in this ontology.

First, a robot is a *Device* (Figure 5):

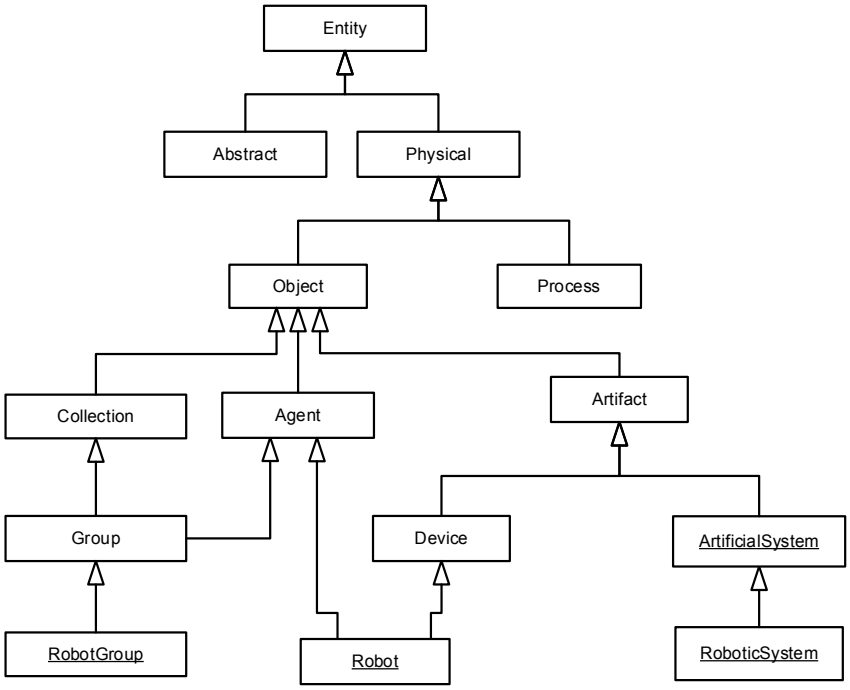

Figure 5—Taxonomy of the main concepts in CORA and their relation to the other ontologies.

```
(subclass Robot Device)
```

SUMO defines a *Device* as an artifact with the purpose of serving as an instrument in a subclass of process. In certain cases, such as those involving multipurpose robots, the class of process may not be known *a priori*.

Instances of Robot can exhibit characteristics through two relations that are indirectly inherited from Object. The attribute relation allows the robot to get qualitative characteristics by instantiating the abstract class *Attribute*. The *Measure* relation allows the robot to get quantitative characteristics instantiating the abstract class PhysicalMeasure.

A Robot is also an *Agent* (see Figure 5).

```
(subclass Robot Agent)
```

Robots perform tasks by acting on the environment or themselves.

7.2 robotPart

Robots have other devices as parts. There are a myriad of devices that can play the role of robot part, and it is not possible to determine in advance what devices can or cannot be robot parts.

Devices that are considered robot parts while attached to a robot are not necessarily always a robot part in an ontological sense, since they exist by themselves and, in most cases, they can be connected to other kinds of devices. For instance, a power source is essentially a device. However, a specific instance of power source can be dynamically considered as a *robotPart*, during a specific time interval, while connected to a robot. This is represented through a subrelation of *part,* which relates devices and robots.

```
(subrelation robotPart part)
(domain robotPart 1 Device)
(domain robotPart 2 Robot)
```

7.3 RobotInterface

A robot interacts with the world surrounding it through an interface.

```
(subclass RobotInterface Device)

(=>
    (instance ?ROBOT Robot)
    (exists (?INTER)
        (and
            (instance ?INTER RobotInterface)
            (robotPart ?INTER ?ROBOT))))
```

The *RobotInterface* is a device composed by other devices that play the roles of sensing device, actuating device and communicating device. Through the interface, the robot can sense and act on the environment as well as communicate with other agents. Therefore, the robot interface can be viewed as way to refer to all the devices that allow the robot to interact with the world.

Every sensing part of a given robot is part of its interface.

```
(forall (?SENSEP)
    (=>
         (robotSensingPart ?SENSEP ?ROBOT)
         (exists (?INTER)
             (and
                 (instance ?INTER RobotInterface)
                 (robotPart ?INTER ?ROBOT)
                 (part ?SENSEP ?INTERFACE)))))
```

Every actuating part of a given robot is part of its interface.

```
(forall (?ACTP)
    (=>
         (robotActuatingPart ?ACTP ?ROBOT)
         (exists (?INTER)
             (and
                 (instance ?INTER RobotInterface)
                 (robotPart ?INTER ?ROBOT)
                 (part ?ACTP ?INTERFACE)))))
```

Every communicating part of a given robot is part of its interface.

```
(forall (?COMMP)
    (=>
         (robotCommunicatingPart ?COMMP ?ROBOT)
         (exists (?INTER)
             (and
                 (instance ?INTER RobotInterface)
                 (robotPart ?INTER ?ROBOT)
                 (part ?COMMP ?INTERFACE)))))
```

Every robot interface must have a part that is either a robot sensing part, or a robot actuating part or a robot communicating part.

```
(forall (?INTER ?ROBOT)
    (=>
        (and
            (instance ?INTER RobotInterface)
            (instance ?ROBOT Robot)
            (robotPart ?INTER ?ROBOT))
        (or
            (exists (?SENSEP)
                (and
                    (robotSensingPart ?SENSEP ?ROBOT)
                    (part ?SENSEP ?INTERFACE)))
            (exists (?ACTP)
                (and
                    (robotActuatingPart ?ACTP ?ROBOT)
                    (part ?ACTP ?INTERFACE)))
            (exists (?COMMP)
                (and
                    (robotCommunicatingPart ?COMMP ?ROBOT)
                    (part ?COMMP ?INTERFACE)))))))
```

Each robot has one and only one robot interface.

```
(forall (?ROBOT ?INTER1 ?INTER2)
    (=>
        (and
            (instance ?ROBOT Robot)
            (instance ?INTER1 RobotInterface)
            (instance ?INTER2 RobotInterface)
            (robotPart ?INTER1 ?ROBOT)
            (robotPart ?INTER2 ?ROBOT))
        (equal ?INTER1 ?INTER2)))
```

7.4 fullyAutonomousRobot , semiAutonomousRobot, teleoperatedRobot, remoteControlledRobot and automatedRobot

The notion that a robot is an agent might raise some questions in situations where a robot requires more input commands from an operator in order to execute tasks; e.g., teleoperated robots. This is one of the main points of ambiguity when differentiating robots from other machines. This issue is related to the problem of defining and representing *autonomy*.

Huang, Albus, and Messina define Autonomy Levels for Unmanned Systems (ALFUS) as one way to evaluate autonomy with respect to robotic devices. CORA imports the notion of "Modes of Operation for Unmanned Systems" defined in ALFUS and redefines it in relation to the concept of process present in SUMO.

CORA assumes that autonomy relates to a particular occurrence of a process. A robot participating as an agent in a process might participate as a fully autonomous robot, as a semi-autonomous robot, as a teleoperated robot, as a remote controlled robot, or as an automated robot. CORA defines these different situations as case roles that a robot can assume in a process.

```
(subrelation agentRobot agent)
(domain 1 agentRobot Robot)
(domain 2 agentRobot Process)

(subrelation fullyautonomousRobot agentRobot)
(subrelation semiautonomousRobot agentRobot)
(subrelation teleoperatedRobot agentRobot)
(subrelation remotecontrolledRobot agentRobot)
(subrelation automatedRobot agentRobot)
```

The relations *fullyAutonomousRobot*, *semiAutonomousRobot*, *teleoperatedRobot*, and *remoteControlledRobot* represent robots participating as agents in processes in the corresponding operation modes defined in ALFUS, which are *fully autonomous, semi-autonomous, teleoperation,* and *remote control*, respectively. CORA complements this list with the role *automatedRobot* attributed to robots acting as automatons in a process, like clockwork, incapable of altering their actions.

It is important to note that a given robot can play multiple roles in different processes at the same time. For example, a robotic rover exploring a planet can assume the semi-autonomous role in the process of planet exploration, but it can be, at the same time, fully autonomous in the process of navigation.

At this stage, CORA does not define any structural or capability restrictions on the robots that assume these different roles.

7.5 RobotGroup

A robot is an agent, and agents can form social groups. According to SUMO, a *group* is "a collection of agents," like a pack of animals, a society, or an organization.

A *RobotGroup* is a group whose only members are robots (Figure 6).

```
(subclass RobotGroup Group)

(forall ?MEMBER
    (=>
        (and
            (instance ?GROUP RobotGroup)
            (member ?MEMBER ?GROUP))
        (instance ?MEMBER Robot)))
```

According to SUMO, a *Group* is an agent, in the sense that it can act on its own. The agents that compose a group establish its agency.

Robot teams, such as robot football teams or teams of soldering robots in factories, are examples of robot groups.

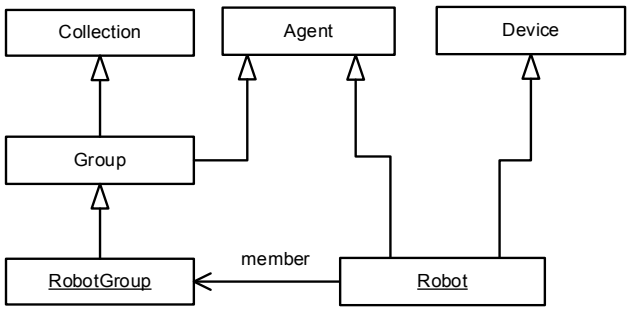

Figure 6—Part of the ontology showing the main concepts related to RobotGroup.

The term *RobotGroup* also includes *complex robots*. These are embodied mechanisms formed by many agents attached to each other. One example is a robotic tank in which the hull and the turret are independent autonomous robots that can coordinate their actions to achieve a common goal. A robotic snake, composed of smaller, autonomous robots, is also an example of complex robot.

7.6 RoboticSystem

Robots and other devices can form a *RoboticSystem* (Figure 7).

A *RoboticSystem* is an artificial system formed by robots and devices intended to support the robots to carry on their tasks (see ISO 8373:2012).

```
(subclass RoboticSystem ArtificialSystem)

(instance support BinaryRelation)
(domain support 1 Device)
(domain support 2 RoboticSystem)
```

```
(=>
     (instance ?SYSTEM RoboticSystem)
     (exists (?ROBOT)
         (and
             (instance ?ROBOT Robot)
             (part ?ROBOT ?SYSTEM))))

(=>
     (instance ?SYSTEM RoboticSystem)
     (exists (?DEVICE)
         (and
             (instance ?DEVICE Device)
             (part ?DEVICE ?SYSTEM)
             (support ?DEVICE ?SYSTEM)
             (not (instance ?DEVICE Robot)))))
```

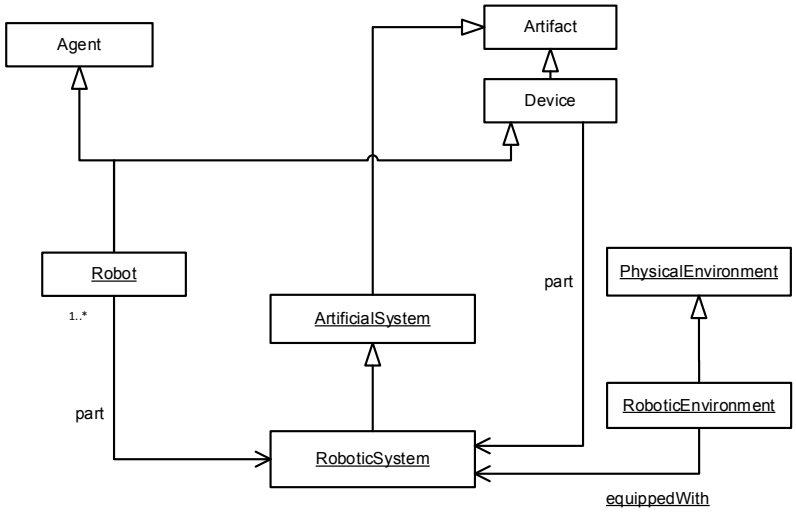

Figure 7— Robotic system and its relations with robot and robotic environment.

7.7 RoboticEnvironment

A physical environment equipped with robotics systems is a *RoboticEnvironment*.

```
(subclass RoboticEnvironment PhysicalEnvironment)

(instance equippedWith BinaryRelation)
(instance equippedWith AsymmetricRelation)
(domain equippedWith 1 RoboticEnvironment)
(domain equippedWith 2 RoboticSystem)


(=>
    (instance ?ENV RoboticEnvironment)
    (exists (?SYSTEM)
        (and
             (instance ?SYSTEM RoboticSystem)
             (equippedWith ?ENV ?SYSTEM))))
```

At least one part of the system that equips a *RoboticEnvironment* must be located in a region in the environment.

```
(=>
    (equippedWith ?ENV ?SYSTEM)
    (exists (?DEVICE ?REGION)
        (and
            (part ?DEVICE ?SYSTEM)
            (part ?REGION ?ENV)
            (located ?DEVICE ?REGION))))
```

7.8 SingleRoboticSystem and CollectiveRoboticSystem

Robotic systems might have only one or more than one robot. Robotic systems are partitioned into *SingleRobotic System*s and *CollectiveRoboticSystem*s (Figure 8).

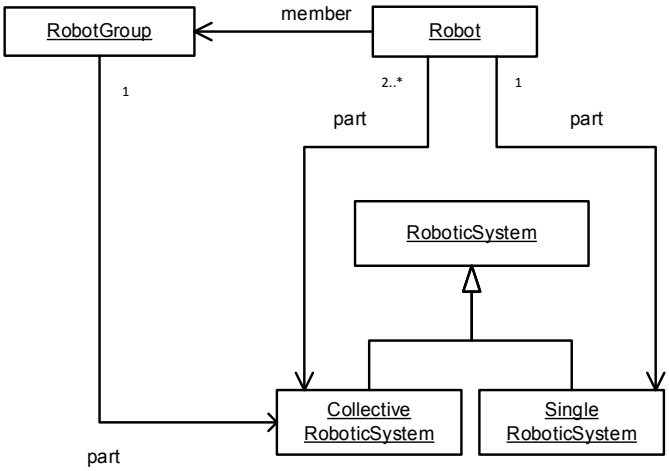

Figure 8—Different types of robotic systems.

```
(subclass SingleRoboticSystem RoboticSystem)
(subclass CollectiveRoboticSystem RoboticSystem)
(partition RoboticSystem SingleRoboticSystem CollectiveRoboticSystem)
```

A *SingleRoboticSystem* has one and only one robot:

```
(=>
    (instance ?SYSTEM SingleRoboticSystem)
    (forall (?RA ?RB)
        (=>
            (and
                (instance ?RA Robot)
                (part ?RA ?SYSTEM)
                (instance ?RB Robot)
                (part ?RB ?SYSTEM))
             (equal ?RA ?RB))))
```

A *CollectiveRoboticSystem* has two or more robots:

```
(=>
    (instance ?SYSTEM CollectiveRoboticSystem)
    (exists (?RA ?RB)
        (and
            (instance ?RA Robot)
            (part ?RA ?SYSTEM)
            (instance ?RB Robot)
            (part ?RB ?SYSTEM)
            (not (equal ?RA ?RB)))))
```

It is assumed that all robots in a *CollectiveRoboticSystem* collaborate in some way to achieve a common goal. Naturally, each robot may have its own local goal, but these goals must be subgoals of the larger one, i.e., the group's goal. As such, all robots in a *CollectiveRoboticSystem* must be members of a single robot group. This is the case even in situations where there is no direct interaction between robots, such as in an automated assembling line.

```
(=>
    (and
        (instance ?SYSTEM CollectiveRoboticSystem)
        (instance ?ROBOT Robot)
        (part ?ROBOT ?SYSTEM))
    (exists (?G1)
        (forall (?G2)
            (=>
                (and
                    (instance ?G1 RobotGroup)
                    (member ?ROBOT ?G1)
                    (instance ?G2 RobotGroup)
                    (member ?ROBOT ?G2))
                (equal ?G1 ?G2)))))
```

In this case, the robot group is also part of the robotic system.

```
(=>
    (and
        (instance ?SYSTEM CollectiveRoboticSystem)
        (instance ?ROBOT Robot)
        (part ?ROBOT ?SYSTEM))
    (forall (?G)
        (=>
            (and
                (instance ?G RobotGroup)
                (member ?ROBOT ?G))
            (part ?G ?SYSTEM))))
```

Another example of robotic system is an automated home assistant system composed by a helper robot and stand-alone sensors and actuators to open doors. In particular, the notion of robotic system is motivated by the need to describe industrial robotic settings in other R&A ontologies.

8. RPARTS axioms

8.1 Background

CORA states that any device can be used as a robot part. In this sense, a robot part is considered a *role* played by some device when it is attached to a robot. The RPARTS ontology aggregates some of the most general and typical specific types of robot parts.

RPARTS provides only a set of specific types of roles that specialize the general role of robot part, defining the requirements that have to be met for a device playing the role of each specific kind of robot part. Nevertheless, this set of roles is extensible. This ontology does not describe the actual devices that can play the role of robot part.

8.2 robotSensingPart

Measuring devices can play the role of a sensing part of a robot when they are connected to robots. For example, a given instance of laser sensor can play the role of *robotSensingPart* when it is connected to a robot.

```
(subrelation robotSensingPart robotPart)
(domain robotSensingPart 1 MeasuringDevice)
(domain robotSensingPart 2 Robot)
```

For example, the situation where a robot *robot1* has a laser sensor *ls1* playing the role of a sensing part can be represented as follows:

```
(subclass LaserSensor MeasuringDevice)


(instance ls1 LaserSensor)
(instance robot1 Robot)
(robotSensigPart ls1 robot1)
```

8.3 robotActuatingPart

Devices that allow the robot to move and act in the surrounding environment can play the role of *robotActuating Part*. For example, a given instance of end effector can play the role of *robotActuating Part* when it is connected to a robot.

```
(subrelation robotActuatingPart robotPart)
(domain robotActuatingPart 1 Device)
(domain robotActuatingPart 2 Robot)

(=>
    (robotActuatingPart ?DEVICE ?ROBOT)
    (exists (?PROC)
        (and
            (subclass ?PROC RobotMotion)
            (hasPurpose ?DEVICE
                (exists (?INST)
                    (and
                        (instance ?INST ?PROC)
                        (instrument ?INST ?DEVICE)))))))
```

For example, the situation where a robot *robot1* has an end effector *ef1* playing the role of an actuating part can be represented as follows:

```
(subclass EndEffector Device)

(instance ef1 EndEffector)
(instance robot1 Robot)
(robotActuatingPart ef1 robot1)
```

8.4 robotCommunicatingPart

The *robotCommunicatingPart* is the role played by any device that serves as an instrument in a robot-robot or human-robot communication process by allowing the robot to send (or receive) information to (or from) a robot or a human. For example, a given instance of a radio receiver can play the role of the *robotCcommunicatingPart* when it is connected to a robot.

```
(subrelation robotCommunicatingPart robotPart)
(domain robotCommunicatingPart 1 Device)
(domain robotCommunicatingPart 2 Robot)

(=>
    (robotCommunicatingPart ?DEVICE ?ROBOT)
    (exists (?PROC)
        (and
            (hasPurpose ?DEVICE
                (exists (?INST)
                    (and
                        (instance ?INST ?PROC)
                        (instrument ?INST ?DEVICE))))
            (or
                (subclass ?PROC HumanRobotCommunication)
                (subclass ?PROC RobotRobotCommunication)))))
```

For example, the situation where a robot *robot1* has a radio receiver *rc1* playing the role of a communicating part can be represented as follows:

```
(subclass RadioReceiver Device)


(instance rc1 RadioReceiver)
(instance robot1 Robot)
(robotCommunicatingPart rc1 robot1)
```

8.5 robotProcessingPart

Processing devices that allow the robot to process information play the role of the *robotProcessingPart*. For example, an instance of controller can play the role of robot processing part when it is connected to a robot.

```
(subclass robotProcessingPart robotPart)
(domain robotProcessingPart 1 ProcessingDevice)
(domain robotProcessingPart 2 Robot)
```

For example, the situation where a robot *robot1* has a controller *c1* playing the role of a processing part can be represented as follows:

```
(subclass Controller Device)


(instance c1 Controller)
(instance robot1 Robot)
(robotProcessingPart c1 robot1)
```

9. POS axioms

9.1 Background

POS is an ontology about position, orientation and pose. Figure 9 depicts the general structure of concepts representing position.

9.2 PositionCoordinateSystem

A *PositionCoordinateSystem* is an abstract entity used for specifying location in relation to a reference object.

```
(subclass PositionCoodinateSystem Abstract)
```

A position coordinate system has a single reference object.

```
(instance refPCS SingleValuedRelation)
(instance refPCS BinaryPredicate)
(domain refPCS 1 PositionCoordinateSystem)
(domain refPCS 2 Object)

(=>
    (instance ?CS PositionCoodinateSystem)
    (exists (?OBJ)
        (refPCS ?CS ?OBJ)))
```

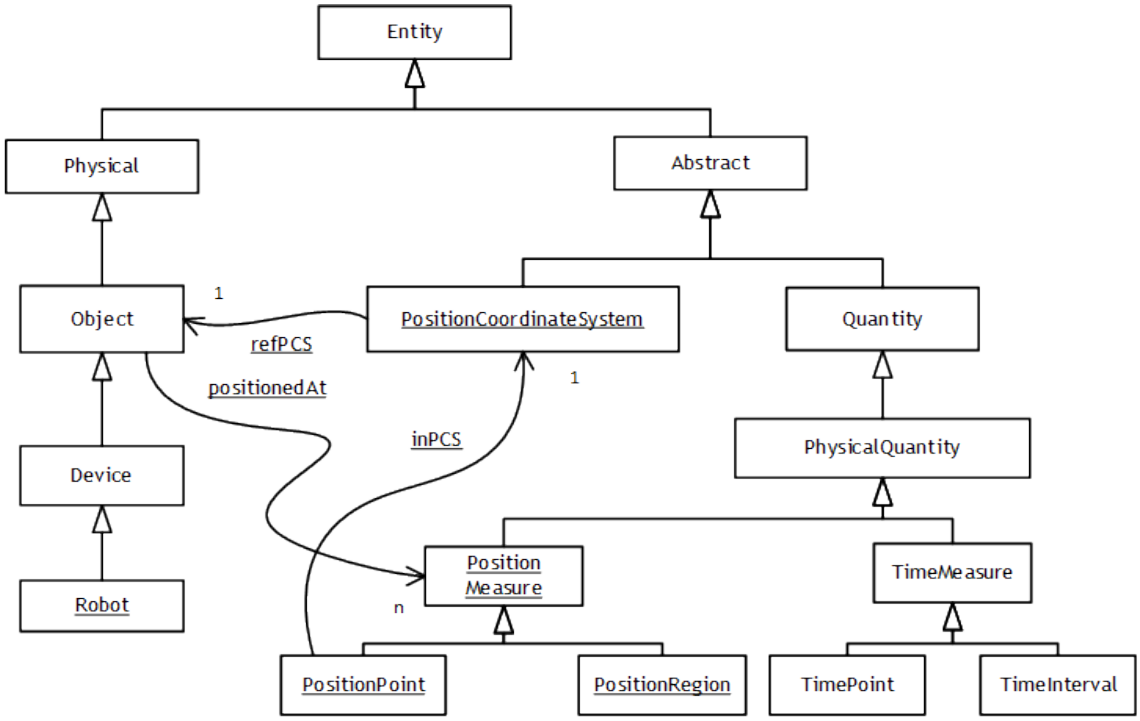

Figure 9—Main concepts in POS ontology regarding Position.

9.3 PositionMeasure

A (spatial) position is essentially a measure attributed to a given (physical) object describing its position. The domain of values from which to take position values is a specialization of physical quantity, as defined by SUMO.

```
(subclass PositionMeasure PhysicalQuantity)
```

A relation between objects and *PositionMeasure* values determines a position.

```
(subrelation positionedAt measure)
(domain positionedAt 1 Object)
(domain positionedAt 2 PositionMeasure)
```

POS divides a *PositionMeasure* into position points and position regions.

```
(partition PositionMeasure PositionPoint PositionRegion)
```

A *PositionMeasure* is always defined according to a single position coordinate system.

```
(instance inPCS SingleValuedRelation)
(instance inPCS BinaryPredicate)
(domain inPCS 1 PositionMeasure)
(domain inPCS 2 PositionCoordinateSystem)
```

```
(=>
    (instance ?P PositionMeasure)
    (exists (?CS)
        (and
            (instance ?CS  PositionCoodinateSystem)
            (inPCS ?P ?CS))))
```

A reference object of a position coordinate system has a position in this coordinate system. This position cannot change.

```
(=>
   (and
       (instance ?O Object)
       (refPCS ?CS ?O)
       (inPCS ?P ?CS)
       (positionedAt ?O ?P))
   (holdsDuring (WhenFn ?O) (positionedAt ?O, ?P)))
```

The relative position of an object in reference to another can be made explicit by means of a *TernaryPredicate*:

```
(instance relPosition TernaryPredicate)
(domain relPosition 1 Object)
(domain relPosition 2 PositionMeasure)
(domain relPosition 3 Object)

(<=>
    (relPosition ?O ?P ?OR)
    (and
        (positionedAt ?O ?P)
        (exists (?CS)
            (and
                (instance ?CS PositionCoordinateSystem)
                (inPCS ?P ?CS)
                (refPCS ?CS ?OR)))))
```

9.4 PositionPoint

A *PositionPoint* denotes the quantitative position of an object in a coordinate system projected on the physical space. A *PositionPoint* is a precise indication of position of a given object.

```
(subclass PositionPoint PositionMeasure)
```

9.5 PositionTransformation

A *PositionTransformation* is a function that maps position points into other position points (possibly in another coordinate system). Position transformations form a *UnaryFunction* class:

```
(subclass PositionTransformationFunction UnaryFunction)

(=>
    (instance ?T PositionTransformationFunction)
    (and
        (domain ?T 1 PositionPoint)
        (range ?T PositionPoint)))
```

There is a mapping from one coordinate system to another if and only if there is a transformation that maps all points in one coordinate system to another.

```
(instance mapsPCS TernaryRelation)
(domain mapsPCS 1 PositionTransformationFunction)
(domain mapsPCS 2 PositionCoordinateSystem)
(domain mapsPCS 3 PositionCoordinateSystem)

 (<=>
    (mapsPCS ?T ?C1 ?C2)
    (forall (?P)
        (=>
            (and
                (instance ?P PositionPoint)
                (inPCS ?P ?C1))
            (exists (?PM)
                (and
                    (inPCS ?PM ?C2)
                    (equal ?PM
                        (?T ?P)))))))
```

Transformations can be composed transitively by the function pCompose.

```
(instance pCompose SingleValuedRelation)
(instance pCompose TernaryPredicate)
(domain pCompose 1 PositionTransformationFunction)
(domain pCompose 2 PositionTransformationFunction)
(domain pCompose 3 PositionTransformationFunction)

(=>
    (pCompose ?T1 ?T2 ?TT)
    (forall (?P ?PM ?PF)
        (=>
            (and
                (equal ?PM
                     (?T1 ?P))
                (equal ?PF
                     (?T2 ?PM)))
            (equal ?PF
                (?TT ?P)))))
```

In robotics (as in other disciplines), coordinate systems are also related through hierarchies (i.e., trees). Usually, an agent chooses a coordinate system as the global reference frame, which constitutes the global coordinate system (GCS) for that agent. This GCS can be arbitrarily chosen and does not have reference a particular coordinate frame. Local coordinate systems (LCS) are defined in relation to GCS by hierarchical links. This notion of hierarchy is arbitrary and defined by the agent and can be represented formally using:

```
(instance parentPCS BinaryPredicate)
(instance parentPCS TransitiveRelation)
(domain parentPCS 1 CoordinateSystem)
(domain parentPCS 2 CoordinateSystem)
```

If two coordinate systems are hierarchically related, all points from one can be mapped to points in the other using transformations. This hierarchical link is represented as:

```
(=>
    (parentPCS ?C1 ?C2)
    (exists (?T1 ?T2)
        (and
            (instance ?T1 PositionTransformationFunction)
            (instance ?T2 PositionTransformationFunction)
            (mapsPCS ?T1 ?C1 ?C2)
            (mapsPCS ?T2 ?C2 ?C1))))
```

This means that if two coordinate systems share a parent node in the hierarchy tree, there is a transformation between them. In fact, this transformation can be built in the following manner:

```
(=>
    (and
        (parentPCS ?C2 ?C1)
        (parentPCS ?C2 ?C3))
    (exists (?T1 ?T2 ?T3)
        (and
          (instance ?T1 PositionTransformationFunction)
          (mapsPCS ?T1 ?C1 ?C2)
          (instance ?T2 PositionTransformationFunction)
          (mapsPCS ?T2 ?C2 ?C3)
          (pCompose ?T1 ?T2 ?T3)
          (mapsPCS ?T3 ?C1 ?C3))))
```

These properties of mappings among coordinate systems are preserved regardless the mathematical formalism that is used.

9.6 PositionRegion

A *PositionRegion* is a qualitative position of an object. It constitutes an abstract region in a coordinate system overlapping the physical spatial region occupied by the object.

```
(subclass PositionRegion PositionMeasure)
```

Position points can be in position regions.

```
(instance inPR BinaryRelation)
(domain inPR 1 PositionMeasure)
(domain inPR 2 PositionRegion)
```

All position points within a *PositionRegion* and the *PositionRegion* itself are in the same coordinate system.

```
(=>
    (instance ?R PositionRegion)
    (exists (?C)
        (and
            (instance ?C PositionCoordinateSystem)
            (forall (?P)
                (=>
                   (inPR ?P ?R)
                   (inPCS ?P ?C))))))
```

POS defines operators that act as a generator of a *PositionRegion* when applied to objects. Operators can be of many types. For instance, a single operator can get the position point of an object and generate a *PositionRegion* that corresponds to the left side of that object.

```
(subclass PositionSpatialOperator Function)

(instance pGenerated SingleValuedRelation)
(instance pGenerated TernaryPredicate)
(domain pGenerated 1 PositionSpatialOperator)
(domain pGenerated 2 List)
(domain pGenerated 3 PositionCoordinateSystem)
(domain pGenerated 4 PositionRegion)
```

The operator must be applied to at least one object.

```
(=>
    (pGenerated ?OP ?LI ?CS ?R)
    (not
        (equal ?LI NullList)))
```

When an operator is applied to a list of objects and a coordinate system, all objects must have a position in that coordinate system.

```
(=>
    (pGenerated ?OP ?LI ?CS ?R)
    (forall (?OBJ)
        (=>
            (inList ?OBJ ?LI)
            (exists (?POS)
                (and
                    (positionedAt ?OBJ ?POS)
                    (inPCS ?POS ?CS))))))
```

The generated region (i.e., all of its points) must be in the coordinate system used to generate it.

```
(=>
    (instance ?R PositionRegion)
    (exists (?OP ?LI ?CS ?R)
        (and
            (instance ?OP PositionSpatialOperator)
            (pGenerated ?OP ?LI ?CS ?R)
            (forall (?P)
               (=>
                 (inPR ?P ?R)
                 (inPCS ?P ?CS))))))
```

The object extension in a coordinate system is the minimal region containing the object in a given coordinate system.

```
(instance extensionPos TernaryPredicate)
(instance extensionPos SingleValuedRelation)
(domain extensionPos 1 Object)
(domain extensionPos 2 PositionCoordinageSystem)
(domain extensionPos 3 PositionRegion)
```

A *PositionRegion* is related by special spatial relations defined in coordinate systems (rather than in physical space). This standard defines at least one relation.

```
(instance overlapsPosition BinaryRelation)
(instance overlapsPosition ReflexiveRelation)
(instance overlapsPosition SymmetricRelation)
(domain overlapsPosition 1 PositionRegion)
(domain overlapsPosition 2 PositionRegion)
```

An object is at a particular *PositionRegion* if the object extension projected on the appropriate coordinate system overlaps with the position region. These notions allow the extension of the definition of position relationship in terms of a *PositionRegion* as follows:

```
(=>
    (and
        (instance ?O Object)
        (instance ?R PositionRegion)
        (positionedAt ?O ?R))
    (exists (?C)
        (and
          (instance ?C PositionCoordinateSystem)
          (extensionPos ?O ?C ?PR)
          (overlapsPosition ?PR ?R))))
```

Relative position can also be defined for a *PositionRegion*.

```
(<=>
    (and
        (relPosition ?O ?R ?OR)
        (instance ?R PositionRegion))
    (exists (?G ?LI ?CS)
        (and
            (positionedAt ?O ?R)
            (pGenerated ?G ?LI ?CS ?R)
            (inList ?OR ?LI)
            (refPCS ?CS ?OR))))
```

9.7 OrientationCoordinateSystem

An *OrientationCoordinateSystem* is an abstract entity.

```
(subclass OrientationCoodinateSystem Abstract)
```

An *OrientationCoordinateSystem* is defined in relation to a single reference object.

```
(instance refOCS SingleValuedRelation)
(instance refOCS BinaryPredicate)
(domain refOCS 1 OrientationCoordinateSystem)
(domain refOCS 2 Object)

(=>
    (instance ?CS OrientationCoodinateSystem)
    (exists (?OBJ)
        (refOCS ?CS ?OBJ)))
```

9.8 OrientationMeasure

Analogous to a position measure, an *OrientationMeasure* is essentially a measure (or observation) attributed to the orientation of a given (physical) object. In this context, orientation denotes the information regarding where the object is pointing to in relation to the reference object of the orientation coordinate system. For instance, the compass heading of an unmanned aerial vehicle is an *OrientationMeasure* in the one-dimensional, circular orientation coordinate system of the compass. The reference object in this case is planet Earth.

The domain of values from which to take orientation values is a specialization of physical quantity.

```
(subclass OrientationMeasure PhysicalQuantity)
```

An *OrientationMeasure* is attributed to objects through the relation *orientated at*.

```
(subrelation orientedAt measure)
(domain orientedAt 1 Object)
(domain orientedAt 2 OrientationMeasure)
```

POS divides an *OrientationMeasure* into orientation value and orientation region.

```
(partition OrientationMeasure
     OrientationValue OrientationRegion)
```

An *OrientationMeasure* is always in a single *OrientationCoordinateSystem*.

```
(instance inOCS SingleValuedRelation)
(instance inOCS BinaryPredicate)
(domain inOCS 1 OrientationValue)
(domain inOCS 2 OrientationCoordinateSystem)

(=>
    (instance ?P OrientationValue)
    (exists (?C)
        (and
            (instance ?C  OrientationCoodinateSystem)
            (inOCS ?P ?C))))
```

Examples of *OrientationMeasure*s vary from simple orientation angles to vectors and other mathematical representations. The type of *OrientationCoordinateSystem* determines what kind of orientation values the coordinate system has. For instance, it is possible to specialize an orientation coordinate system in order to define a two-dimensional vector orientation system, in which the orientation values are two-dimensional unit vectors.

The relative orientation of an object in reference to another can be explicitly stated by means of a *TernaryPredicate*.

```
(instance relOrientation TernaryPredicate)
(domain relOrientation 1 Object)
(domain relOrientation 2 OrientationMeasure)
(domain relOrientation 3 Object)

(<=>
    (relOrientation ?O ?P ?OR)
    (and
        (orientedAt ?O ?P)
```

```
        (exists (?CS)
          (and
             (instance ?CS OrientationCoordinateSystem)
             (inOCS ?P ?CS)
             (refOCS ?CS ?OR)))))
```

9.9 OrientationValue

An *OrientationValue* denotes the quantitative orientation of a given object in an orientation coordinate system. An orientation value is a precise indication of orientation of a given object.

```
(subclass OrientationValue OrientationMeasure)
```

9.10 OrientationTransformation

An *OrientationTransformation* is a function that maps *OrientationValue* to *OrientationValue* (possibly in another *OrientationCoordinateSystem*). Orientation transformation functions form a *UnaryFunction* class:

```
(subclass OrientationTransformationFunction UnaryFunction)

(=>
    (instance ?T OrientationTransformationFunction)
    (and
        (domain ?T 1 OrientationValue)
        (range ?T OrientationValue)))
```

There is a map from an orientation coordinate system to another if and only if there is a transformation that maps all orientation values from the first coordinate system to the second.

```
(instance mapsOCS TernaryRelation)
(domain mapsOCS 1 OrientationTransformationFunction)
(domain mapsOCS 2 OrientationCoordinateSystem)
(domain mapsOCS 3 OrientationCoordinateSystem)

 (<=>
    (mapsOCS ?T ?C1 ?C2)
    (forall (?P)
        (=>
            (and
                (instance ?P OrientationMeasure)
                (inOCS ?P ?C1))
             (exists (?PM)
                (and
                    (inOCS ?PM ?C2)
                        (equal ?PM
                            (?T ?P)))))))
```

An *OrientationTransformation* can be composed transitively by the function *oCompose*.

```
(instance oCompose SingleValuedRelation)
(instance oCompose TernaryPredicate)
(domain oCompose 1 OrientationTransformationFunction)
(domain oCompose 2 OrientationTransformationFunction)
(domain oCompose 3 OrientationTransformationFunction)
```

```
(=>
    (oCompose ?T1 ?T2 ?TT)
    (forall (?P ?PM ?PF)
        (=>
            (and
                (equal ?PM
                    (?T1 ?P))
                (equal ?PF
                    (?T2 ?PM)))
            (equal ?PF
                (?TT ?P)))))
```

An *OrientationCoordinateSystem* can be arranged in a hierarchy.

```
(instance parentOCS TransitiveRelation)
(domain parentOCS 1 OrientationCoordinateSystem)
(domain parentOCS 2 OrientationCoordinateSystem)
```

As with a *PositionCoordinateSystem*, if two *OrientationCoordinateSystem*s are hierarchically related, it is possible to map all points from one to the other using transformations. This hierarchical link is represented as

```
(=>
    (parentOCS ?C1 ?C2)
    (exists (?T1 ?T2)
        (and
            (instance ?T1 OrientationTransformationFunction)
            (instance ?T2 OrientationTransformationFunction)
            (mapsOCS ?T1 ?C1 ?C2)
            (mapsOCS ?T2 ?C2 ?C1))))
```

This means if two coordinate systems share a parent node in the hierarchy tree, then there is a transformation between them. In fact, this transformation is obtained in the following manner

```
(=>
    (and
        (parentOCS ?C2 ?C1)
        (parentOCS ?C2 ?C3))
    (exists (?T1 ?T2 ?T3)
        (and
          (instance ?T1 OrientationTransformationFunction)
          (mapsOCS ?T1 ?C1 ?C2)
          (instance ?T2 OrientationTransformationFunction)
          (mapsOCS ?T2 ?C2 ?C3)
          (oCompose ?T1 ?T2 ?T3)
          (mapsOCS ?T3 ?C1 ?C3))))
```

9.11 OrientationRegion

An *OrientationRegion* is an abstract region used to represent the qualitative orientation in an *OrientationCoordinateSystem*. An *OrientationRegion* is analogous to position regions in the sense that it represents general, usually imprecise information about orientation. For instance, the sentence “the robot is heading west” specifies that the robot is oriented at the *OrientationRegion* west in the compass’ *OrientationCoordinateSystem*. In this case, west is a region in the coordinate system in the sense it encompasses all orientation values generally interpreted as “heading west” (see Figure 10).

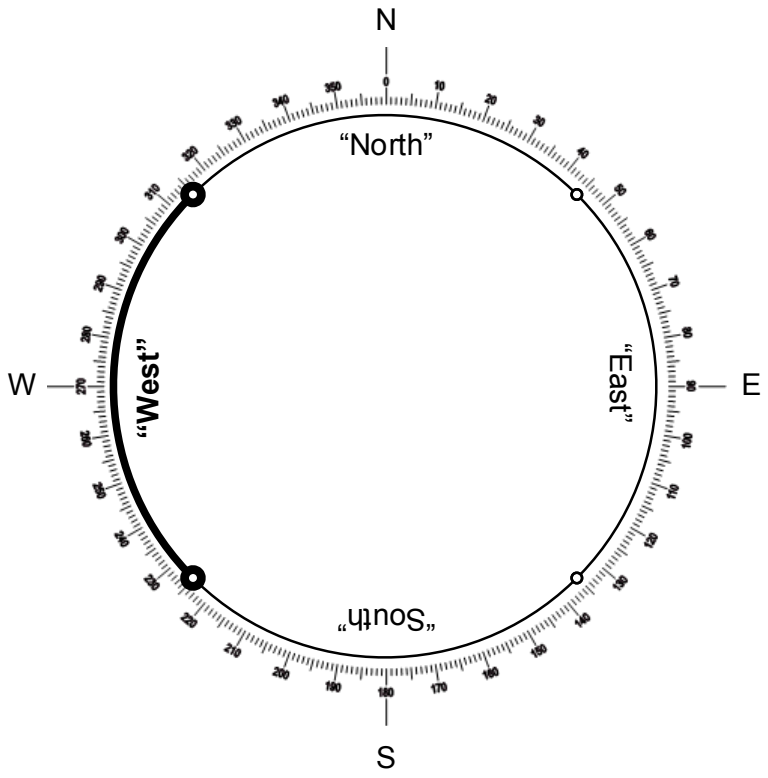

Figure 10 — Orientation measures and regions in the compass orientation coordinate space[9]

An *OrientationRegion* is an *OrientationMeasure*:

```
(subclass OrientationRegion OrientationMeasure)
```

Orientation values can be located inside an *OrientationRegion*.

```
(instance inOPR BinaryRelation)
(domain inOPR 1 OrientationMeasure)
(domain inOPR 2 OrientationRegion)
```

All orientation values within an *OrientationRegion* and the *OrientationRegion* itself are in the same coordinate system.

```
(=>
    (instance ?R OrientationRegion)
    (exists (?C)
        (and
           (instance ?C OrientationCoordinateSystem)
           (forall (?P)
              (=>
                 (inOPR ?P ?R)
                 (inOCS ?P ?C))))))
```

[9] The orientation space represented in Figure 10 is a one-dimensional, circular space. It is divided into four intervals (or regions) corresponding to an *OrientationRegion*. The emphasized region corresponds to the west *OrientationRegion*.

POS defines operators that act as a generator of an *OrientationRegion* when applied to objects. Operators can be of many types. For instance, a single operator can get the orientation value of an object and generate an orientation region that corresponds to the north of that object.

```
(subclass OrientationSpatialOperator Function)

(instance oRGenerated SingleValuedRelation)
(instance oRGenerated TernaryPredicate)
(domain oRGenerated 1 OrientationSpatialOperator)
(domain oRGenerated 2 Object)
(domain oRGenerated 3 OrientationRegion)

(=>
    (instance ?R OrientationRegion)
    (exists (?OR ?G ?C)
        (and
            (instance ?G OrientationSpatialOperator)
            (oRGenerated ?G ?OR ?R)
            (instance ?C OrientationCoordinateSystem)
            (refOCS ?C ?OR)
            (forall (?P)
                (=>
                  (inOPR ?P ?R)
                  (inOCS ?P ?C))))))
```

Relative orientation can also be defined for an *OrientationRegion*:

```
(<=>
    (and
        (relOrientation ?O ?R ?OR)
        (instance ?R OrientationRegion))
    (exists (?G)
        (and
            (orientedAt?O ?R)
            (oRGenerated ?G ?OR ?R))))
```

9.12 Pose

The *Pose* of some object is a quantity comprising orientation and position.

```
(subclass PoseMeasure PhysicalQuantity)

(subrelation pose measure)
(domain 1 pose Object)
(domain 2 pose PoseMeasure)

(instance posePosition SingleValuedRelation)
(instance posePosition BinaryPredicate)
(domain posePosition 1 PoseMeasure)
(domain posePosition 2 PositionMeasure)

(instance poseOrientation SingleValuedRelation)
(instance poseOrientation BinaryPredicate)
(domain poseOrientation 1 PoseMeasure)
(domain poseOrientation 2 OrientationMeasure)
```

A *PoseMeasure* having an *OrientationValue* and a *PositionPoint* is called a *QuantitativePose*.

```
(subclass QuantitativePose PoseMeasure)

(=>
    (and
        (instance ?POSE QuantitativePose)
        (posePosition ?POSE ?P)
        (poseOrientation ?POSE ?O))
    (and
        (instance ?P PositionPoint)
        (instance ?O OrientationValue)))
```

9.13 PoseTransformation

It is possible to define *transformations* between quantitative poses. These transformation form a class of unary functions:

```
(subclass QuantPoseTransformationFunction UnaryFunction)

(=>
    (instance ?T QuantPoseTransformationFunction)
    (and
        (domain ?T 1 QuantitativePose)
        (range ?T QuantitativePose)))
```

A *QuantitativePoseTransformation* relies on the existence of position and orientation transformations between the position and orientation quantities that characterize the pose:

```
(=>
    (and
        (instance ?POSE1 QuantitativePose)
        (instance ?POSE2 QuantitativePose)
        (instance ?T QuantPoseTransformationFunction)
        (equal (?T ?POSE1) ?POSE2))
    (exists (?P1 ?P2 ?O1 ?O2 ?PT ?OT)
        (and
            (posePosition ?POSE1 ?P1)
            (posePosition ?POSE2 ?P2)
            (instance ?PT PositionTransformationFunction)
            (equal (?PT ?P1) ?P2)
            (poseOrientation ?POSE1 ?O1)
            (poseOrientation ?POSE2 ?O2)
            (instance ?OT OrientationTransformationFunction)
            (equal (?OT ?O1) ?O2))))
```

Annex A

(informative)

Ontology: general aspects

In computer science, ontologies are formal tools that enable the description of objects, properties and relationships among such objects in a knowledge domain. In particular, two main definitions capture the essence, purpose and scope of an ontology. Studer et al. [B19][10] combined the definitions by Gruber [B7] and Borst [B1] and states that an ontology is "an explicit, formal specification of a shared conceptualization." On the other hand, Guarino [B8] stresses the formal aspects of a conceptualization and defines ontologies as "logical theories accounting for the intended meaning of a formal vocabulary." An ontology comprises at least a set of terms and their definitions as shared by a given community, formally specified in a machine-readable language, such as first-order logic. Ontologies are particularly important to provide machines with knowledge representation and reasoning capabilities to solve a task, as well as to allow high-level interoperability between systems.

The term ontology encompasses several ways of structuring its elements. From a mere list of terms and definitions to a formal theory specified—for instance, in first order logic—the structure of what has to be modeled changes dramatically at both extremes (see Uschold and Gruninger [B22]). Notwithstanding, the main elements of an ontology can be identified as [B6]:

— *Classes*: stand for concepts at all granularities;

— *Relations*: stand for associations between concepts;

— *Formal axioms*: constrain and add consistency rules to the concept and relationship structures;

The range of activities concerning the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tools and languages that support them is called *ontological engineering* (or *ontology engineering*) [B6].

Disparate classifications are available for systematizing different kinds of ontologies. This document adopts the classification of *levels of generality* by Guarino [B8], namely:

— Top-level ontologies, which describe very general concepts (like space, time, matter, object, event, action, etc.), which are independent of a particular problem or domain

— Domain ontologies, which describe concepts of a specific domain

— Task ontologies, which describe generic tasks or activities

— Application ontologies, which are strictly related to a specific application and used to describe concepts of a particular domain and task

It is important to notice that there is a reusability-usability trade-off regarding the application of ontologies [B6], i.e., general ontologies are more reusable and less usable than specific ontologies. This trade-off implies that by going down in the above classification, towards a greater specificity, there is an increase of usability and a decrease in reusability of the ontologies.

[10] The numbers in brackets correspond to those of the bibliography in Annex C.

This document presented a *core ontology*. Not listed in the classification above, core ontologies can be viewed as mid-level ontologies, positioned between top-level ontologies and domain ontologies [B17]. Core ontologies reuse concepts defined by top-level ontologies and specify new concepts that can be used in particular domains and tasks. Core ontologies specify concepts that are general in a large domain, such as—in the R&A domain—*robot*, *device*, and *robotic system* and their corresponding relationships.

Annex B

(informative)

Ontology development

B.1 Background

The development of CORA is supported by two well-known methodologies for building ontologies: METHONTOLOGY [B4] and OntoClean [B9]. METHONTOLOGY includes the identification of the ontology development process, a life cycle based on evolving prototypes, and particular techniques for carrying out each activity. The ontology development process refers to *which* activities are carried out when building ontologies. It includes three categories of activities that must be performed: project management activities, development-oriented activities, and support activities.

METHONTOLOGY includes the following *project management activities*:

— Planning, which identifies which tasks will be performed, how they will be arranged, how much time and what resources (such as other ontologies) are needed for their completion

— Control, which helps guarantee that planned tasks are completed in the manner that they were intended to be performed

— Quality assurance, which helps to assure that the quality of each and every resulting product (ontology, software and documentation) is satisfactory

METHONTOLOGY also includes the following *development-oriented activities*:

— Specification, which defines the purpose and scope of the ontology, its intended usage, and target users. This activity also specifies some sources that could be used to acquire knowledge for the ontology development.

— Conceptualization, which organizes and converts an informally perceived view of a domain into a semi-formal specification using a set of *intermediate representations* [B2] based on tabular and graph notations that can be understood by domain experts and ontology developers. The result of the conceptualization activity is the ontology conceptual model.

— Formalization, which transforms the conceptual model into a formal or semi-computable model. This can be done, for example, by specifying a model using first order logic.

— Implementation, which transforms the ontology previously formalized into a computable model, codified in an ontology representation language, such as OWL.

— Maintenance, which updates and corrects the ontology.

Finally, METHONTOLOGY specifies the following *support activities*, which can be performed continuously throughout the development process, and simultaneously with the *development-oriented activities*:

— Knowledge acquisition, which directs the acquisition of domain knowledge from several sources, such as domain experts, domain literature (books and papers), other ontologies, thesauri, domain

glossaries, etc. This activity also includes the employment of techniques such as brainstorming, interviews, formal and informal analysis of texts, knowledge acquisition tools, etc.

— Evaluation, which helps evaluate the technical quality of the ontology during and in between each activity. The ORA WG has adopted OntoClean [B9] as the main methodology of evaluation, which can be used along all the processes, checking for the quality of the taxonomies in the ontology. Furthermore, the ORA WG used the analysis of counter-examples discussed by Sure et al. [B20] as a useful approach to evaluate the definitions of ontology concepts. It can help to determine when an expected instance of a class is not correctly classified due to problems in the definition of the class. Once the ontology is implemented in some codification language (such as OWL), several tools can be used to evaluate it. Among these tools, reasoners are important, since they support several types of evaluations, such as: consistency checking, subsumption checking, equivalence checking and instantiation checking. In this activity, ORA WG includes other steps of evaluation in which a subgroup submits its partial results to other subgroups as well as to the R&A community. Other useful ontology evaluation approaches can be viewed in the work of Hartmann et al. [B10]. It is important to note that an evaluation can identify several kinds of problems, some of which may require referral to previous activities in order to be solved.

— Integration, the goal of which is to consider the reuse of definitions already built into other ontologies. This is the case, for example, when a definition given in an upper-level ontology (as DOLCE or SUMO) is adopted, or when a specific domain ontology extends a concept given in the core ontology.

— Documentation, which helps ensure that the development process and the ontology itself are documented. This activity includes, in the broad sense, the maintenance and tracking of the sources of the terms and definitions of the ontology, as well as the clear definitions and examples embedded in the code of the implemented ontology.

— Configuration Management, which records all versions of the documentation, software, and ontology code to control the changes.

In METHONTOLOGY [B4], the *ontology life cycle* identifies the set of stages through which the ontology moves during its lifetime and describes what activities could be performed in each stage and how the stages are related (relation of precedence, return, etc.). In the evolving prototype life cycle, the ontology grows depending on the needs. This model allows for modification, addition, and removal of definitions in the ontology at any time.

In the ontology life cycle proposed by METHONTOLOGY [B4], project management activities are performed during the whole ontology development process. The ontology life cycle moves forward through the following states: specification, conceptualization, formalization, integration, implementation and maintenance. In any of these states, the process can move towards the first state (specification), restarting the cycle. The support activities (knowledge acquisition, evaluation of ontologies, and documentation) are carried out during the whole life of the ontology.

B.2 Development activities in CORA

The UpOM Working subgroup began its work by specifying the sources from which the domain knowledge could be acquired. The main sources are existing standards in the domain, textbooks, peer-reviewed papers, domain experts, and other ontologies (including upper ontologies as DOLCE and SUMO). Moreover, the UpOM group also decided to adopt a *middle-out approach* proposed by Uschold et al. [B23] for identifying concepts, namely starting from the most relevant concepts and branching out both to the most abstract and to the most concrete ones.

Among the existing ontologies, the upper-level ontologies are very important, since they provide higher-level concepts that support the definition of concepts in CORA. On the other hand, the ontologies for R&A collected from literature encompass only a subset of terms of R&A, with specialized meaning for specific applications. Moreover, they have been developed by a small group of people, not representing the common shared knowledge of the R&A community.

The UpOM Working subgroup started working with the ISO/TC184/SC2 committee (Robotics and robotic devices) Working Group 1 (Vocabulary). This working group published ISO 8373, which defines (in natural language) generic terms that are common in R&A. This document was considered as a good starting point for developing the glossary of terms that would form the ontology, since its development moved the community towards broad agreement. The section “General Terms” was the most promising aspect for the development of CORA. It contains terms that describe the most general notions across the sub-domains of industrial robots and service robots. It helped in clarifing the places where there is disagreement and the places where there is agreement about terminology. A subset of terms has been chosen to be included in CORA, such as robot, autonomy, robot system, robotic device, and so on. However, in most cases, the definitions of some terms were changed.

From the terms identified in the previous activities, the conceptualization activity has been performed. In this step, the domain knowledge was organized using the intermediate representations discussed by Corcho et al. [B2], as suggested by METHONTOLOGY [B4], for representing the main concepts, taxonomies, and relationships among the concepts. Furthermore, unified-modeling language (UML) diagrams were used for representing this knowledge in a format that facilitated the overview of the main concepts, relations and taxonomies proposed by Gómez-Pérez et al. [B6]. ISO 8373 was remarkably valuable in this project for identifying some of the core terms of R&A and for making the structure of the domain knowledge explicit.

The results acquired in the previous step motivated further analysis of the literature in order to identify alternative conceptualizations of the core notions of the ontology. In this analysis, several alternative definitions for *robot* were found, which emphasized different aspects of the term under different perspectives. Moreover, according to some works in the literature, some core notions in the R&A domain, such as *robot* [B3] and *automation* [B16] are surprisingly hard to define. Joseph Engelberger, a pioneer in R&A domain, expressed these difficulties in his comment: “I can't define a robot, but I know one when I see one” [B13]. All this evidence indicates that accommodating all the alternative conceptualizations the community has about robots under a single broad definition is a challenging task.

Alternative definitions of robot found in the literature highlighted some important aspects of this concept, including the ideas of robot as agent [B18], robot as a programmable machine with actuators and sensors [B5], [B12], [B14], and robot as having capability of performing a variety of tasks [B5], [B12], [B21]. The main terms used in CORA were defined based on these aspects.

In the next step, the ORA WG decided to integrate CORA with an upper-level ontology. Upper-level ontologies with clear ontological commitments can help the modeler to make consistent choices based on well-founded premises. If everyone working on distinct lower-level ontologies commits to the same top-level ontology, it is easier to integrate the lower-level ontologies into the larger framework. Moreover, according to Jansen and Schulz [B11] the commitment to an upper-level ontology avoids incorrect conclusions and mistakes typical of ad-hoc approaches for modeling ontologies, forcing the ontology developers to think about ambiguous terms and providing them a more precise definition. CORA is aligned with SUMO, which is the most prominent proposal under consideration by the IEEE Standard Upper Ontology [B15]. SUMO provides a good description of the top-most categories, includes the main notions and distinctions introduced in CORA and allows a broader interpretation of notions such as agent, device, and agent group. CORA takes this material and incorporates the notion of autonomy introduced in the ALFUS framework. According to ALFUS, autonomy is determined by the contextual autonomous capability model, which specifies that autonomy depends on three axes, namely: mission complexity, environmental complexity and human independence.

Annex C

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] Borst, W. N., "Construction of Engineering Ontologies for Knowledge Sharing and Reuse." *Ph.D. thes.*, Universiteit Twente, 1997.[11]

[B2] Corcho, O., M. Fernandez-López, A. Gómez-Pérez, and A. López-Cima, "Building legal ontologies with METHONTOLOGY and WebODE," *Law and the Semantic Web: Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications. Lecture Notes in Computer Science*, vol. 3369, Springer, pp. 142–157, 2005.[12]

[B3] Craig, J. J., *Introduction to Robotics: Mechanics and Control,* Third Ed. Prentice Hall, 2005.

[B4] Fernández, M., A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: from ontological art towards ontological engineering," *AAAI Spring Symposium*, USA, pp. 33–40, Mar. 1997.[13]

[B5] Fu, K. S., R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision and Intelligence.* McGraw-Hill, 1987.

[B6] Gómez-Pérez, A. and V. R. Benjamins, "Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods," *in: Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods*, Stockholm, Sweden, pp. 1–15, 1999.[14]

[B7] Gruber, T. R., "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, no. 5–6, pp. 907–928, 1995.

[B8] Guarino, N., "Formal Ontology in Information Systems," *Proceedings of the International Conference on Formal Ontology in Information Systems*, Trento, Italy, pp. 3–15, 1998.

[B9] Guarino, N. and C. A. Welty, "An overview of OntoClean," *Handbook on Ontologies.* Springer, pp. 201–220, 2009.[15]

[B10] Hartmann, J., P. Spyns, A. Giboin, D. Maynard, R. Cuel, M. C. Surez-Figueroa, and Y. Sure, "D.1.2.3. Methods for ontology evaluation," Technical Report, EU-IST Network of Excellence (NoE) Knowledge Web Consortium, 2005.[16]

[B11] Jansen, L. and S. Schulz, "The ten commandments of ontological engineering," *Proceedings of the Workshop Ontologies in Biomedicine and Life Science*, Berlin, Germany, pp. 41–46, 2011.

[B12] Lund, H. H., "Modular robotics for playful physiotherapy," *Proceedings of the IEEE International Conference on Rehabilitation Robotics*, Kyoto, Japan, pp. 571–575, 2009.[17]

[B13] Macura, K. J. and D. Stoianovici, "Advancements in magnetic resonance-guided robotic interventions in the prostate," *Topics in Magnetic Resonance Imaging,* vol. 19, no. 6, pp. 297–304, Dec. 2008.[18]

[11] Available at: http://doc.utwente.nl/17864/
[12] Available at: http://www.springer.com/us/book/9783540250630
[13] Available at: http://aaaipress.org/Papers/Symposia/Spring/1997/SS-97-06/SS97-06-005.pdf
[14] Available at: http://ceur-ws.org/Vol-18/1-gomez.pdf
[15] Available at: http://link.springer.com/chapter/10.1007%2F978-3-540-92673-3_9
[16] Available at: http://www.starlab.vub.ac.be/research/projects/knowledgeweb/KWeb-Del-1.2.3-Revised-v1.3.1.pdf
[17] IEEE publications are available from The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[B14] Munich, M. E., Ostrowski, J. and Pirjanian, P., "ERSP: a software platform and architecture for the service robotics industry," *Proceedings of the International Conference on Robots and Systems*, Alberta, Canada, pp. 460–467, 2005.

[B15] Niles, I. and A. Pease, "Origins of the IEEE Standard Upper Ontology," *Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, pp. 4-10, 2001.

[B16] Nof, S. Y., *Springer Handbook of Automation*, Springer-Verlag, 2009.

[B17] Obrst, L., "Ontological Architectures," *Theory and Applications of Ontology: Computer Applications*, Springer, pp.27–66, 2010.

[B18] Steels, L. "When are robots intelligent autonomous agents," *Robotics and Autonomous Systems*, vol. 15, pp. 3–9, 1995.[19]

[B19] Studer, R., V. R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," *Data and Knowledge Engineering*, vol. 25, no. 1–2, pp. 161–197, 1998.[20]

[B20] Stabb, S, A Gómez-Pérez, and N. F. Noy, "Why evaluate ontology technologies? Because it works!," *IEEE Intelligent Systems*, vol. 19, no. 4, pp. 74–81, 2004.

[B21] Thrun, S., "Toward a framework for human-robot interaction," *Human–Computer Interaction*, vol. 19, no. 1–2, pp. 9–24, 2004.[21]

[B22] Uschold, M. and M. Gruninger, "Ontologies and semantics for seamless connectivity," *SIGMOD Record*, vol. 33, no. 4, pp. 58–64, 2004.[22]

[B23] Uschold, M., and M. King, "Towards a methodology for building ontologies," *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing.*[23]

[18] Available at: http://journals.lww.com/topicsinmri/pages/articleviewer.aspx?year=2008&issue=12000&article=00006&type=abstract
[19] Available at: http://www.sciencedirect.com/science/article/pii/0921889095000114
[20] Available at: http://www.sciencedirect.com/science?_ob=ArticleListURL&_method=list&_ArticleListID=-742938880&_sort=r&_st=13&view=c&md5=b1e5c351fe6b77108e3b80e4e5d98e11&searchtype=a
[21] Available at: http://www.tandfonline.com/doi/abs/10.1080/07370024.2004.9667338?journalCode=hhci20#.VPYpSvnF-So
[22] Available at: http://www.sigmod.org/publications/sigmod-record/0412/12.uschold-9.pdf
[23] Available at: http://www.aiai.ed.ac.uk/project/pub/documents/1995/95-ont-ijcai95-ont-method.ps

Consensus

WE BUILD IT.

Connect with us on:

Facebook: https://www.facebook.com/ieeesa

Twitter: @ieeesa

LinkedIn: http://www.linkedin.com/groups/IEEESA-Official-IEEE-Standards-Association-1791118

IEEE-SA Standards Insight blog: http://standardsinsight.com

YouTube: IEEE-SA Channel

IEEE
standards.ieee.org
Phone: +1 732 981 0060 Fax: +1 732 562 1571