IEEE SA
STANDARDS
ASSOCIATION

IEEE Standard for Robot Task Representation

IEEE Robotics and Automation Society

Developed by the
Standing Committee for Standards

IEEE Std 1872.1™-2024

STANDARDS

IEEE

IEEE Standard for Robot Task Representation

Developed by the

Standing Committee for Standards
of the
IEEE Robotics and Automation Society

Approved 15 February 2024

IEEE SA Standards Board

Abstract: Defined in this standard is an ontology that allows for the representation of, reasoning about, and communication of task knowledge in the learning, robotics, and automation domain. This ontology includes a list of essential terms and their definitions, attributes, types, structures, properties, constraints, and relationships. In addition, addresses how hierarchical planners and designers represent task knowledge allowing them to better communicate among levels of the ontology hierarchy.

Keywords: IEEE 1872.1, robot knowledge, robot ontology, robotics, tasking

---

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

PDF: ISBN 979-8-8557-0553-9 STD26803
Print: ISBN 979-8-8557-0554-6 STDPD26803

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (https://standards.ieee.org/ipr/disclaimers.html), appear in all standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents."

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

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA 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 makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement IEEE Standards documents do not guarantee safety, security, health, or environmental protection, or guarantee against interference with or from other devices or networks. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. 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 their 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: THE NEED TO PROCURE 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 balloting 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 is 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, nor 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 the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s).

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, 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 subcommittees of the IEEE SA Board of Governors 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 evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the IEEE SA myProject system.[1] An IEEE Account is needed to access the application.

Comments on standards should be submitted using the Contact Us form.[2]

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 constitute 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.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

[1]Available at: https://development.standards.ieee.org/myproject-web/public/view.html#landing.
[2]Available at: https://standards.ieee.org/content/ieee-standards/en/about/contact/index.html.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; https://www.copyright.com/. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

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 10 years. When a document is more than 10 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 IEEE Xplore or contact IEEE.[3] For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE SA Website.[4] Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in IEEE Xplore. Users are encouraged to periodically check for errata.

Patents

IEEE standards are developed in compliance with the IEEE SA Patent Policy.[5]

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

[3]Available at: https://ieeexplore.ieee.org/browse/standards/collection/ieee.
[4]Available at: https://standards.ieee.org/standard/index.html.
[5]Available at: https://standards.ieee.org/about/sasb/patcom/materials.html.

filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at https://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.

IMPORTANT NOTICE

Technologies, application of technologies, and recommended procedures in various industries evolve over time. The IEEE standards development process allows participants to review developments in industries, technologies, and practices, and to determine what, if any, updates should be made to the IEEE standard. During this evolution, the technologies and recommendations in IEEE standards may be implemented in ways not foreseen during the standard's development. IEEE standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

Participants

At the time this draft standard was completed, the P1872.1 Robot Task Ontology Working Group had the following membership:

Stephen Balakirsky, *Chair*
Craig Schlenoff, *Vice Chair*
Signe Redfield, *Secretary*
Patrick Martin, *Ontology Architect*

Julita Bermejo-Alonso
Abdelghani Chibani
Nak Young Chong
Anthony Downs
Brian Fogelson
Antonios Gasteratos
Paulo Jorge Sequeira Gonçalves
Maki K. Habib
Tamas Haidegger
Daniel P. Ireland
Zeid Kootbally
Sitar Kortik
Junbin Liu
Fatima Sabiu Maikore
Jacek Malek
Jeffrey V. Mosley
Hirenkumar Nakawala
Ricardo Sanz
David Scheidt

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

Boon Chong Ang
Ruby Annette
Stephen Balakirsky
Yasen Cai
Robin Chen
Abdelghani Chibani
Diego Chiozzi
Anthony Downs
Nicola Fabiano
A. Ferraro
Brian Fogelson
Paulo Jorge Sequeira Gonçalves
Haiping Guo
Maki K. Habib
Tamas Haidegger
Werner Hoelzl
Masao Ito
Steve Jones
Piotr Karocki
Quist-Aphetsi Kester
Tomasz Piotr Kucner
Mikhail Lagoda
T. Leopold
Howard Li
Jun Li
Dash Liu
Fatima Sabiu Maikore
Jeffrey V. Mosley
Hirenkumar Nakawala
Shahram Negari
Joanna Olszewska
Bansi Patel
Bhushan Jayeshkumar Patel
George Percivall
Davy Pissoort
Cam Posani
Venkatesha Prasad
Edson Prestes
Joao Quintas
S. Veera Ragavan
RK Rannow
Signe Redfield
Ricardo Sanz
Craig Schlenoff
Jhony Sembiring
Taeyoung Uhm
Karl Weber
Yi Yuan
Peng Zhang

When the IEEE SA Standards Board approved this standard on 15 February 2024, it had the following membership:

David J. Law, *Chair*
Vacant Position, *Vice Chair*
Gary Hoffman, *Past Chair*
Alpesh Shah, *Secretary*

Sara R. Biyabani
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Guido R. Hiertz
Ronald W. Hotchkiss
Hao Hu
Yousef Kimiagar
Joseph L. Koepfinger*
Howard Li
Xiaohui Liu
John Haiying Lu
Kevin W. Lu
Hiroshi Mano
Paul Nikolich
Robby Robson
Jon Walter Rosdahl
Mark Siira
Lei Wang
F. Keith Waters
Sha Wei
Philip B. Winston
Don Wright

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1872.1-2024, IEEE Standard for Robot Task Representation.

Ever since the word *robot* was translated to English (from the Czech word *robota* meaning "servitude, forced labor") in 1923, it has become difficult to define a robot without using the word task. For example, in IEEE Std 1872™-2015, a robot is defined as an agentive device in a broad sense, purposed to act in the physical world to accomplish one or more tasks.

Task remains an essential concept based on which robot missions are assumed, irrespective of whether a single robot or a swarm work independently, collaboratively, or cooperatively. Informally, a task is a unit of work that needs to be accomplished through some action. As the design requirements of a robot system increase, so does the complexity of the tasks it can achieve and the corresponding task definitions.

This standard defines an ontology for task representation, where *task* refers to the job the robot is attempting to do rather than the action, behavior, or capability it uses to perform it. If the task consists of a single action by one agent, then it is an *atomic task*. *Composite tasks* are assembled from multiple subtasks and can be performed by a single robotic agent or allocated to various robotic agents to perform collectively. To further represent tasks, different concepts need to be organized and structured in different frames, such as *command frame*, *task composition frame*, *results frame*, *constraints frame*, and *resources frame*.

IEEE Std 1872-2015 (CORA—Core Ontology for Robotics and Automation) was created to enable additional ontology-based standards for robotics and automation, such as this one. The standard defined concepts that were generic to all robotic domains. As such, further specialization was made possible under the CORA framework. When the CORA working group was formed and its participants expanded, several sub-groups emerged and were organized in several layers. The top one, CORA, developed IEEE Std 1872-2015. The middle layer group that developed this standard (IEEE Std 1872.1™) is related to task representation. Other lower layer groups were formed, such as Autonomous Robots (IEEE Std 1872.2™-2021).

Conceptualizing tasks and other related aspects are made in this standard using ontologies that formally structure the concepts and relationships of the domain and characterize task-related knowledge. This ontology provides a common understanding that can be shared by all stakeholders involved, that includes robot manufacturers, system integrators, robot end-users, robot equipment suppliers, other standardization groups' experts, robot software developers, and researchers/developers.

Contents

IEEE Standard for Robot Task Representation

1. Overview

1.1 Scope

This standard defines an ontology that allows representation of, reasoning about, and communication of task knowledge in the learning, robotics, and automation domain. This ontology includes a list of essential terms as well as their definitions, attributes, types, structures, properties, constraints, and relationships. In addition, it addresses how hierarchical planners and designers can represent task knowledge, allowing them to better communicate among levels of the ontology hierarchy.

1.2 Purpose

The standard aims at providing a set of well-founded ontologies specifying vocabulary and definitions about shared concepts and relations in Industrial Robotics and Automation. By providing a standard vocabulary and control schema for industrial robots, end-users can utilize systems from different suppliers in their operations. This has the potential to increase competition and reduce prices for these systems.

Human-robot and heterogeneous robot-robot communication and interaction require well-defined, implementation-independent, standard vocabulary and definitions. Such requirements become particularly relevant in industrial contexts where safe robot communication and integration require clearly defined standards. Ontologies constitute a tool to create semantically rich, formal vocabularies. Recent standardization efforts by IEEE are employing ontologies for standardization in Robotics and Automation (R&A). However, such efforts have been limited to general terminology covering the entire field. Individual subdomains of R&A, such as Industrial Robotics, require specific theories, implying the need for standardized ontologies.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).[6,7]

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

[6] The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.
[7] The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

1.4 General

One of the basic requirements for any robot communication (whether with other robots or humans) is a common vocabulary and clear and concise definitions. The first ontological effort in this line was IEEE Std 1872-2015, which was created to support follow-on standards such as this one.

Among the core areas that were needed was the effective representation of robot task knowledge. This standard aims to fill this void. This includes the ability to represent knowledge, such as commands, task composition, outcomes, constraints, and resources, and provides metadata on each element, such as goal, plan, observation, metrics, outcomes, and capability.

With the growing complexity of behaviors that robots are expected to perform and the need for multi-robot and human-robot collaboration, the need for a standard and well-defined task representation is evident. While the details of tasks vary from application to application, the core structure for such knowledge can remain constant. This standard focuses on this structure and the semantics behind the terms used in the structure's design. IEEE Std 1872.1's standard knowledge representation: 1) more precisely defines the concepts in the robot's knowledge representation, 2) ensures common understanding among community members, and 3) facilitates more efficient data integration and transfer of task knowledge among robotic systems.

The stakeholders that benefit from this standard are robot manufacturers, system integrators, robot end-users (part manufacturers, automotive industry, construction industry, services and solution providers, etc.), robot equipment suppliers, other standardization groups experts, robot software developers, and researchers/ developers.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they shall 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.

IEEE Std 1872™-2015, IEEE Standard Ontologies for Robotics and Automation.[8]

W3C, OWL-S: Semantic Markup for Web Services, 2004.[9]

3. Definitions, acronyms, and abbreviations

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.[10]

[8]IEEE publications are available from The Institute of Electrical and Electronics Engineers (https://standards.ieee.org/).
[9]Available at https://www.w3.org/Submission/OWL-S /. Copyright © 2004 France Telecom, Maryland Information and Network Dynamics Lab at the University of Maryland, National Institute of Standards and Technology (NIST), Network Inference, Nokia, SRI International, Stanford University, Toshiba Corporation, and University of Southampton. All Rights Reserved.
[10]*IEEE Standards Dictionary Online* is available at: http://dictionary.ieee.org. An IEEE account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

3.1.1 General terms

The general terms define concepts that describe concepts that sit above or outside the categorization structure.

action: An operation applied by an agent or team to affect a change in or maintain either an agent's state(s), the environment, or both.

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

approval: The stage of task accomplishment where the task is authorized.

approval package: The input to the approval stage of task accomplishment.

approval role: The role taken on by the agent(s) approving the task attempt.

authorization: The output of the approval stage of task accomplishment.

completion: The stage of task accomplishment where the task attempt is ended.

dynamic system: A system where one or more state variables change with time.

environment: All objects/entities, properties, and information relevant to the task that does not include the self or team of agents performing the task.

evaluation: The stage of task accomplishment where the outcome of a task attempt is assessed.

evaluation role: The role taken on by the agent(s) evaluating the outcome of the task.

execution: The stage of task accomplishment where actions are taken as part of a task attempt.

initialization: The stage of task accomplishment where the task attempt is begun.

specification role: The role taken on by the agent(s) using this IEEE 1872.1 ontology to specify the task.

task: The specification of a mission, problem, or goal to undertake and accomplish or solve.

task accomplishment stage: A stage in the fulfillment of the task attempt.

task attempt: An attempt to complete the task.

task execution role: The role taken on by the agent(s) attempting to accomplish the task.

user role: The role taken on by the agent(s) initiating the task attempt and/or monitoring the task's progress.

3.1.2 General form of robot task

The general form terms define the categories of terms used to organize and structure the concepts that define a task.

constraints frame: The frame containing concepts that limit, constrain, or shape the approval, initialization, execution, evaluation, and completion of a task.

command frame: The frame containing concepts that relate to the initiation of a task.

frame: A category of IEEE 1872.1 ontological concepts.

resources frame: The frame containing concepts that can be used to define the details of the approval, initialization, execution, evaluation, and completion of the task.

results frame: The frame containing concepts that relate to the evaluation and completion of a task.

task composition frame: The frame containing concepts that relate to the execution of a task.

3.1.3 Command frame

The command frame is designed to contain information on the task's specific objective or the desired outcome of a task. It contains definitions of goal that describe this outcome, state that provides the means for this description, and parameter that allows for concrete values to be applied to abstract concepts.

goal: The metrics and outcomes that define the desired state.

parameter: A value used to configure or constrain the task.

state: The state variables that fully describe the system and its temporal changes to any given set of inputs.

3.1.4 Task composition frame

The task composition frame (TCF) is designed to contain information on the execution of a task. The TCF contains definitions of plan that describe a list of intended actions, as well as atomic and composite tasks to provide a task structure. The TCF also includes the definitions of exit, faulted exit, and nominal exit that provide the actions taken to stop a task attempt.

atomic task: An irreducible task.

composite task: A task constructed from any combination of one or more tasks but shall not consist of a single atomic task.

exit: The actions taken when a task attempt stops.

faulted exit: One of a set of possible exits triggered by a fault[11] while the task is being executed.

nominal exit: One of a set of possible exits after the task agent has completed the task without experiencing a fault.

plan: A proposed arrangement in advance that considers a list of intended actions associated with timing and resources for doing or achieving a goal in the future.

3.1.5 Results frame

The results frame is designed to contain information on evaluating whether the task accomplished its goal and how well that goal was accomplished. It includes the definitions of the measurement process, which evaluates measures to produce observations, and the assessment process, which evaluates metrics to produce outcomes, which are determined by the agent in the evaluation role to be either a success, failure, or indeterminate.

assessment: The act of evaluating the metrics.

[11] "Fault" is defined as "manifestation of an error in software" or "incorrect step, process, or data definition in a computer program" or "defect in a hardware device or component" in ISO/IEC/IEEE 24765:2017 Systems and software engineering—Vocabulary. Available at: https://www.iso.org/standard/71952.html.

failure: The final outcome where the metric is unsatisfied.

indeterminate: The final outcome where the metric is neither satisfied nor unsatisfied.

measure: A function over observations, state variables, and parameters.

measurement: The act of evaluating the measures.

metric: A function over observations, outcomes, and parameters.

observation: A value produced by a measure.

outcome: A value produced by a metric.

success: The final outcome where the metric is satisfied.

3.1.6 Constraints frame

The constraints frame is designed to capture the information related to the events that drive how the task is performed. A constraint is defined by an evaluation function constructed from an externally provided resource ontology.

constraint: One or more factors that limit, contain, or help shape the execution of the task by the agent.

evaluation function: The comparison used to establish a constraint.

3.1.7 Resources frame

The resources frame is designed to define the entities that are required to specify, perform, or evaluate a task. The resource properties define a resource's accuracy, confidence, completeness, and sufficiency.

resource: Any entity required to specify, perform, or evaluate a task.

resource property: An attribute associated with a resource that is used to determine whether a resource is suitable for a task.

source: A resource that produces a value.

3.2 Relationships between definitions

The definitions and terms put forward in this standard were not designed to stand on their own. The terms are utilized in the task ontology and are connected by various properties, as illustrated in the following figures.

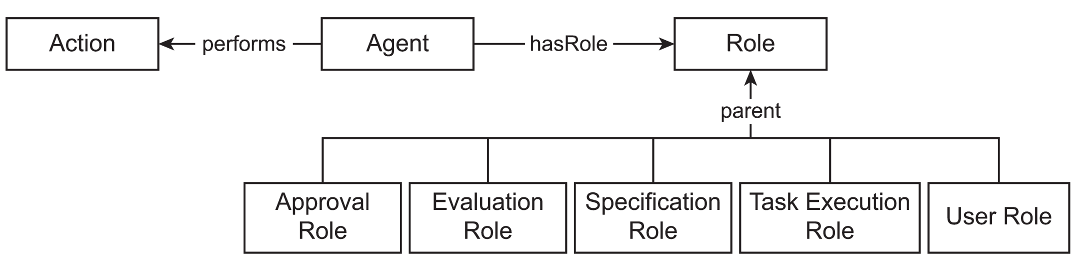

Figure 1—Relationship between various roles

As shown in Figure 1, an *agent* performs one or more *actions* and takes on one or more role types during *task* execution: *approval*, *evaluation*, *specification*, *task execution*, and *user*. These roles are formally defined in 3.1.1. For example, consider an engineer that designs a task for autonomous search and rescue. This agent serves in the *task specification* role but may switch to a *user* role to request that the *task* be executed on a robot. An *agent* in the *task execution* role handles such a request and invokes the necessary steps to execute the *task*. Once the *task* is completed, another *agent* in the *environment* (human or robot) may assume the *evaluation* role to assess whether the *task* completed successfully.

— Agent **performs** one or more Actions and **hasRole** one or more Roles
— Approval **parent** is Role
— Evaluation **parent** is Role
— Specification **parent** is Role
— Task execution **parent** is Role
— User **parent** is Role

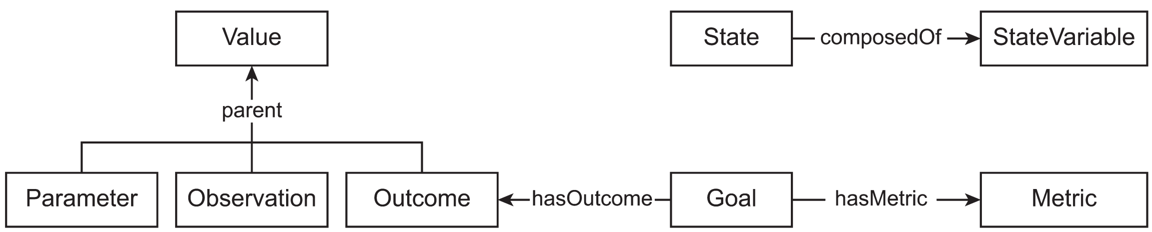

Figure 2—Relationship which enables command frame terms

Figure 2 illustrates the relationships among the *command frame* terms defined in 3.1.3. The entities in this frame specify the objective of the *task*. The core terms defined in this frame are *goal*, *state*, and *parameter*. The remaining terms in this figure come from other frames or are defined outside of the standard, such as *state variable*. Every *goal* must have one or more *metrics* and zero or more *outcomes*. Additionally, the *state* for the task is composed of one or more *state variables*. *Parameters* are used by entities in the *results frame*, which will be discussed next.

— Parameter **parent** is value
— Observation **parent** is value
— Outcome **parent** is value
— Goal **hasOutcome** zero or more outcomes
— Goal **hasMetric** one or more metrics
— State is **composedOf** one or more state variables

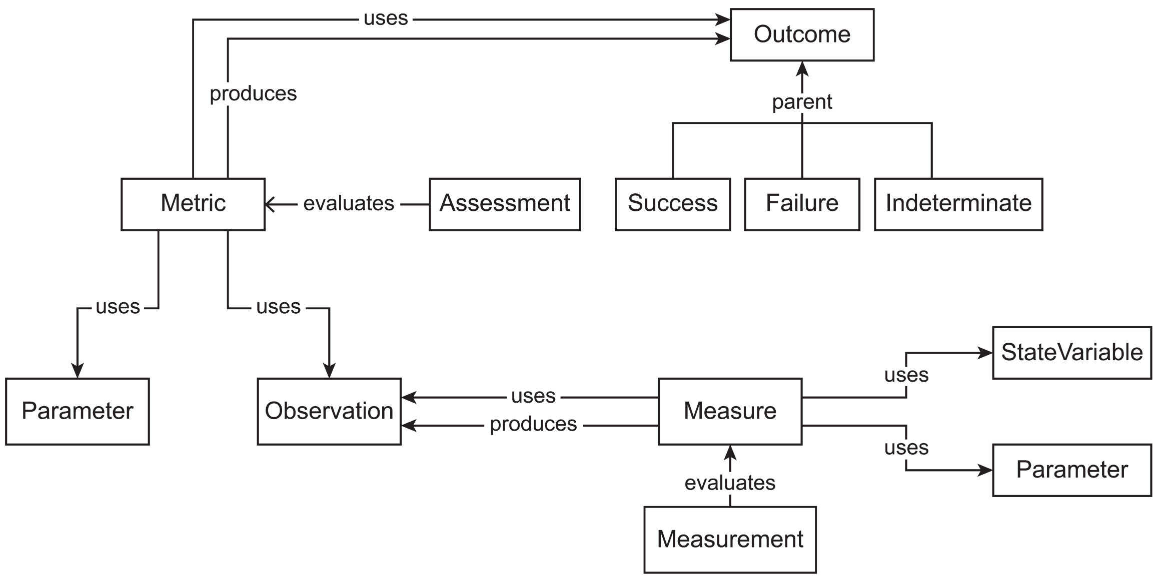

Figure 3—Relationship between various results frame terms

The relationships among the *results frame* and *command frame* entities are shown in Figure 3. The *results frame* has several key terms that support how goals are structured and evaluated: *measure*, *measurement*, *metric*, *assessment*, *observation*, and *outcome*. At the lowest level, the *results frame* facilitates the specification of a *measurement* process, which evaluates *measures*. These *measures* produce *observations* based upon *state variables*, *parameters*, and other *observations*. The *assessment* process evaluates *metrics* that produce *outcomes* based upon *observations*, *parameters*, and other *outcomes*. These *outcomes* may take on one of three subtypes: *success*, *failure*, or *indeterminate*.

— **Metric** *uses* one or more **Outcomes**

— **Metric** *uses* zero or more **Parameters**

— **Metric** *uses* one or more **Observations**

— **Assessment** *evaluates* one or more **Metrics**

— **Metric** *produces* an **Outcome**

— **Success** *parent* is **Outcome**

— **Failure** *parent* is **Outcome**

— **Indeterminate** *parent* is **Outcome**

— **Measure** *produces* an **Observation**

— **Measurement** *evaluates* one or more **Measures**

— **Measure** *uses* zero or more **Observations**

— **Measure** *uses* zero or more **StateVariables**

— **Measure** *uses* zero or more **Parameters**

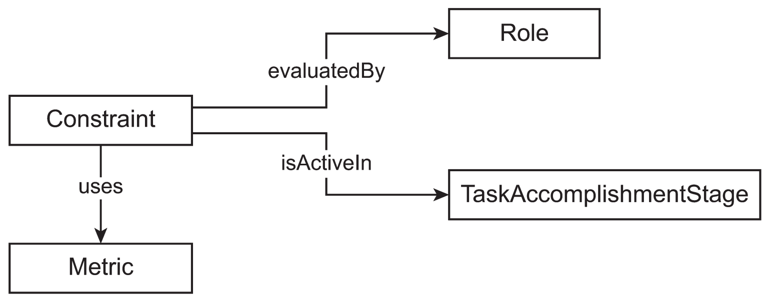

Figure 4—Relationship between various constraints frame terms

Figure 4 illustrates how *constraint* is related to other entities in the standard. First, a *constraint* is constructed from one or more *metrics*, as defined in the *results frame*. These *constraints* are evaluated by an *agent* that has assumed one of the roles shown in Figure 1. Furthermore, *constraints* are active in one or more *task accomplishment stages*.

— **Constraint** *uses* one or more **Metrics**

— **Constraint** is *evaluated* by a **Role**

— **Constraint** is *active* in one or more **TaskAccomplishmentStages**

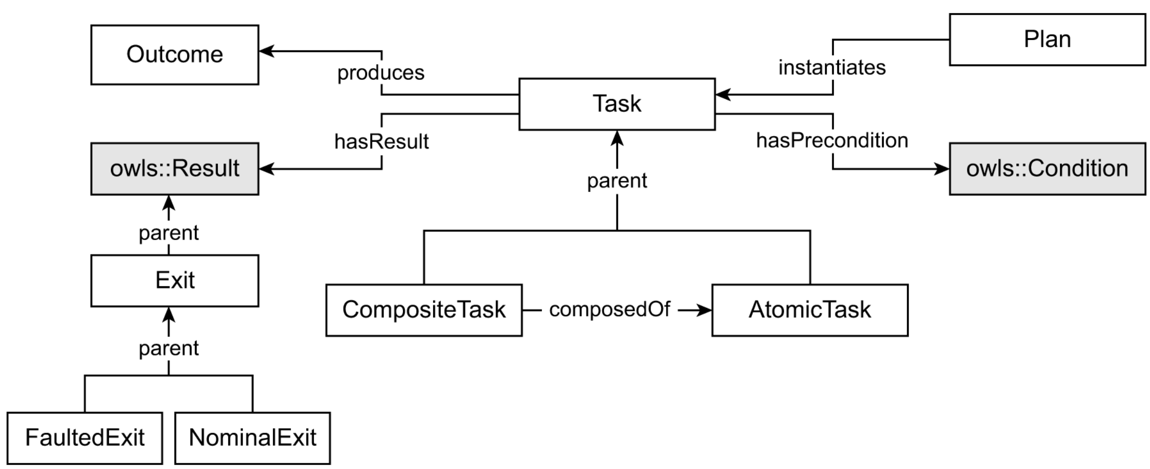

Figure 5—Relationship between task composition frame terms

Figure 5 shows the key relationships among the *task composition frame* entities. At its core, a *task* is an OWL-S process. Please see Clause 2 for more information on OWL-S. By inheriting from this type, a *task* gains a rich set of concepts that supports the specification of *tasks*. Key among these concepts are *composite task* and *atomic task*. An agent in the *specification role* may build a *composite task* from one or more *atomic tasks* using the OWL-S control construct framework, which we abstract with the "composedOf" relation in Figure 5. These control constructs use control structures, such as sequence, if-then-else, or split+join, to compose one or more *atomic tasks* or other *composite tasks*.

Tasks also produce one or more *results*, just like their OWL-S process superclass. The *task composition frame* concept of *exit* is captured as an OWL-S result. At this time, *faulted exit* and *nominal exit* are defined, but additional *exit* types may be added based on user needs. This standard also augments the OWL-S standard by having a *task* produce an *outcome*, as defined in the *command frame*. Also, a *task* is instantiated by a *plan*, which is performed by an agent serving in the *task execution role* with support from agents in *user* and *evaluation roles*.

— **Task** is an ***OWL-S Process***
— **Plan** *instantiates* a **Task**
— **CompositeTask** *parent* is **Task**
— **AtomicTask** *parent* is **Task**
— **CompositeTask** is *composed* of one or more ***AtomicTasks***
— **Task** *produces* an **Outcome**
— **Task** *has* one or more ***OWL-S Results***
— **Exit** *parent* is ***OWL-S Result***
— **FaultedExit** *parent* is **Exit**
— **NominalExit** *parent* is **Exit**

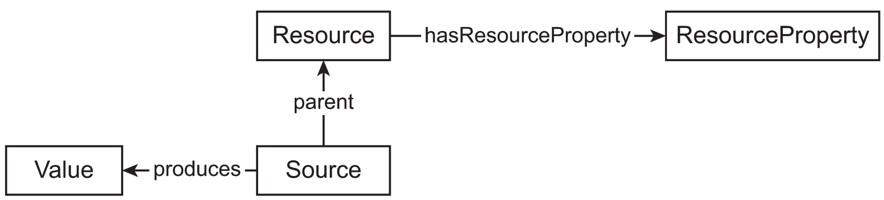

Figure 6—Relationship between resource frame terms

Figure 6 shows the relationships between *resource*, *resource property*, *source*, and *value*. *Resources* can have *resource properties* like accuracy, dimensions, input-information-required, or output-information-produced. Any *source* can be a *resource*, and any *source* can produce one or more *values*. This enables representation of sensors as *resources*, where any given sensor can provide one or more *values*.

— **Resource** *hasResourceProperty* zero or more **ResourceProperties**
— **Source** *parent* is **Resource**
— **Source** *produces* one or more **Values**

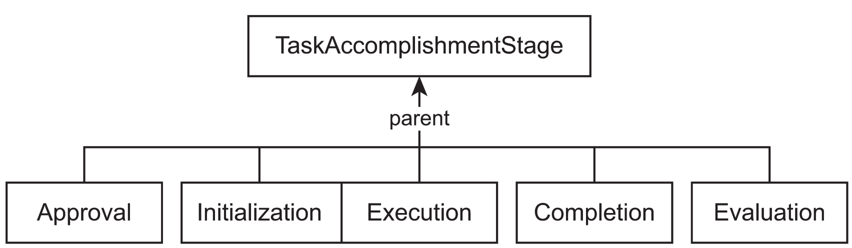

Figure 7—Stages of task accomplishment

Before an attempt to accomplish the *task* may begin, the *task* shall be specified. Once *task specification* is complete, there are five stages of *task accomplishment*. *Approval* includes any necessary external approvals that must be obtained before the *task attempt* can begin. *Initialization* covers activities by the agent in the *task execution role* prior to *task* execution and includes any precondition checks that shall be performed. *Execution* covers the *actions* taken by the agents in the *task execution* and *user roles* during the *task attempt*. *Completion* includes any additional *actions* required to exit from the *task attempt*, and *evaluation* includes assessment activities undertaken by the agent in the *evaluation role* during and/or after the *task attempt*.

— **Approval** *parent* is **TaskAccomplishmentStage**

— **Initialization** *parent* is **TaskAccomplishmentStage**

— **Execution** *parent* is **TaskAccomplishmentStage**

— **Completion** *parent* is **TaskAccomplishmentStage**

— **Evaluation** *parent* is **TaskAccomplishmentStage**

3.3 Usage notes

The purpose of this standard is to create an ontology by defining terminology that identifies the various parts of a *task* and defining the relationships between those terms. The frames provide a mechanism for categorizing the various terms and definitions and are not intended to be used to define the structure of an individual *task*. In practice, this means that *tasks* are defined by combining terms across frames rather than by attempting to force task elements into specific frames. An illustrative example is provided in Annex A.

3.4 Acronyms and abbreviations

CORA	Core Ontology for Robotics and Automation
OWL-S	Web Ontology Language for Web Services
PDDL	Problem Domain Definition Language
R&A	Robotics and Automation
RTR	Robot Task Representation
SC	Standing Committee for Standards
TCF	Task Composition Frame

Annex A

(informative)

Example use case

A.1 Introduction

This annex provides an example of using the standard terms described in this document to implement the hierarchical task of installing an electrical module on a fixture. The use case that this example is based upon is the replacement of an electrical module in an unmanned space station. The system to be tasked utilizes a multi-level architecture for accomplishing this tasking.

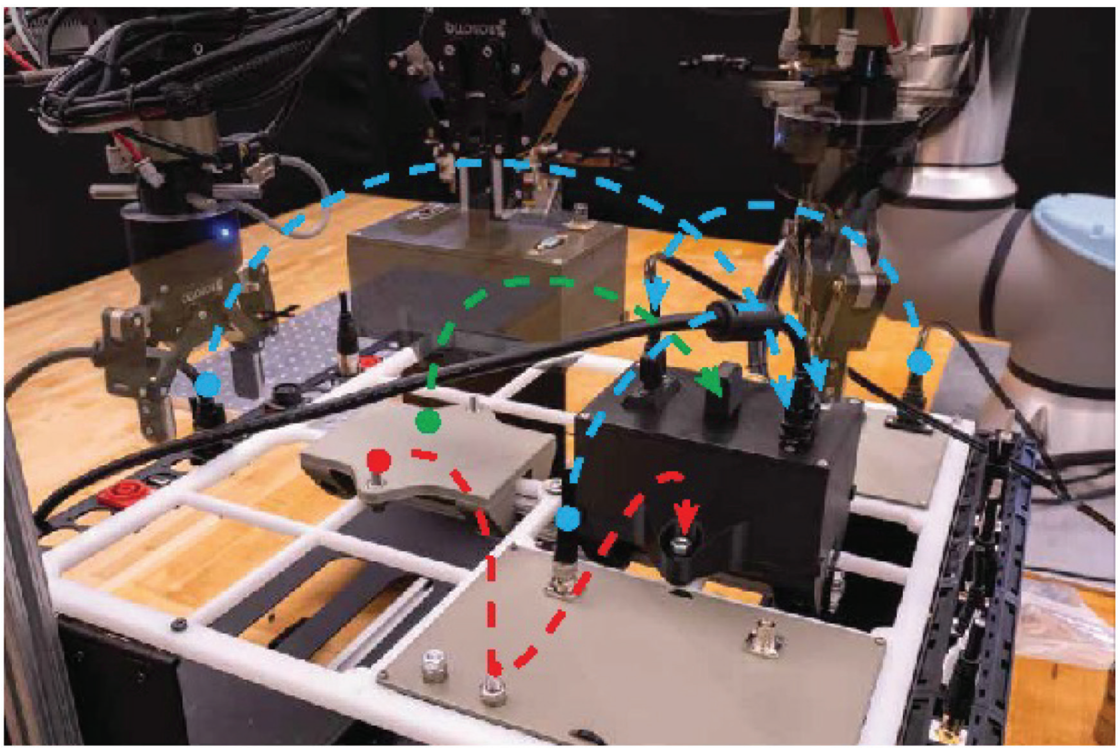

Figure A.1—Task is to complete the installation of an electrical module. The *task* includes moving the module onto a fixture (green arrow), securing the module (red arrow), and wiring various connectors to the module (blue arrows).

As shown in Figure A.1, to be installed, the electrical module must be secured to a fixture and have several electrical connecters connected to the correct locations on the module. The exact module to be installed may be set by the specification. The remaining subclauses of this annex detail how this complex, multi-step installation can be represented in the IEEE 1872.1 framework.

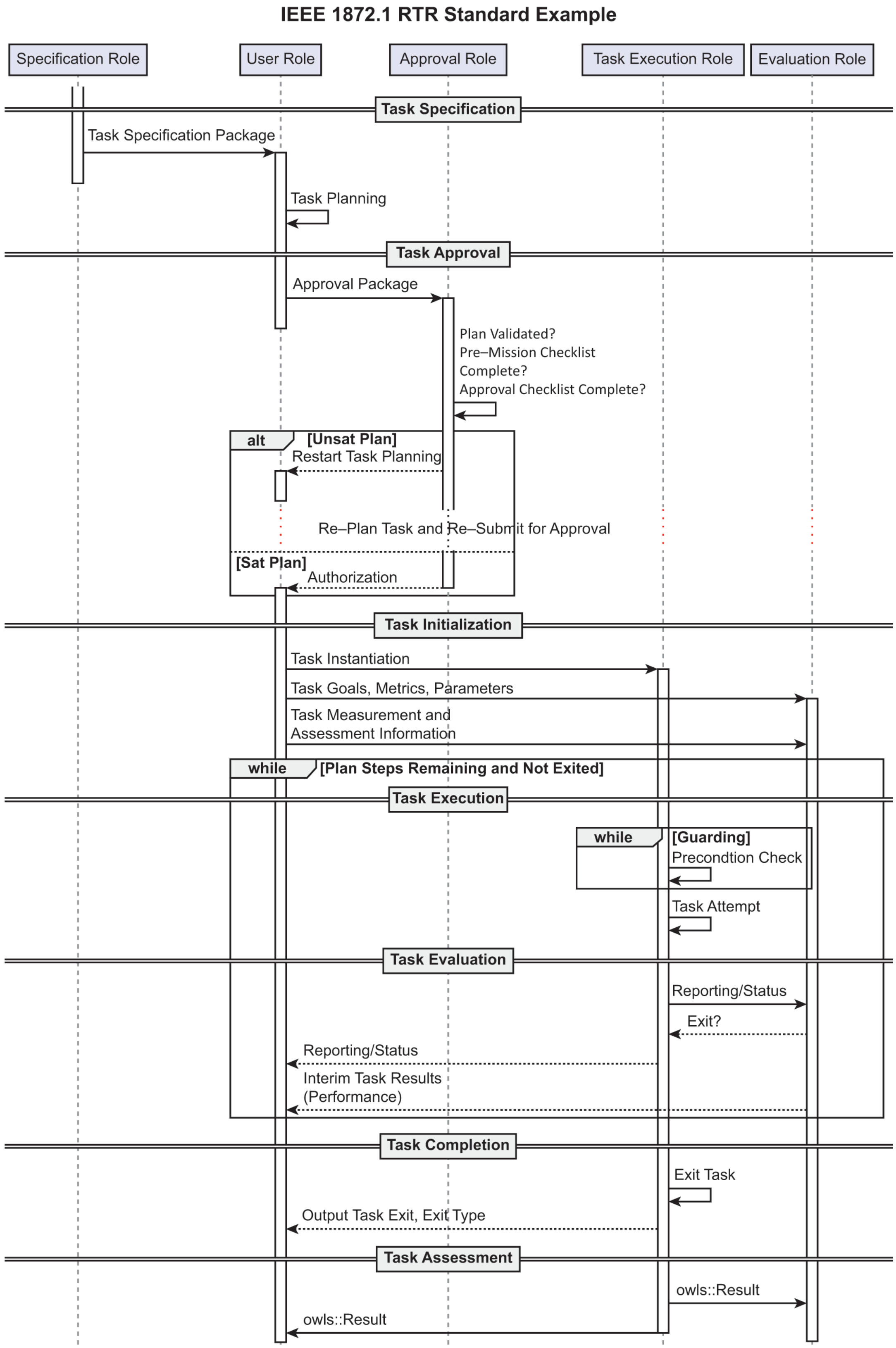

Figure A.2—Various roles that an agent may assume

A.2 Generalized task workflow

As shown in Figure A.2, the overall process starts with *taskspecification*. The agent in the *taskspecification role* sends the agent in the *user role* a task specification package that contains all of the necessary information for the *user role* to understand the *task* to be accomplished, as well as a template for completing a pre-mission checklist. Successful completion of this pre-mission checklist assures that the system is ready and able to complete the proposed *task*.

As further shown in the Figure A.2, the *user role* creates an approval package. This approval package contains the *task plan* and the completed checklist. The *task plan* is an instantiated list of *atomic tasks* that need to be executed. It should be noted that while each step of the plan represents an *atomic task* for this task execution system, when operating in a hierarchical architecture this level's *atomic task* may be a subordinate level's *composite task*. In that case, this entire workflow is repeated by the subordinate system. For the system being examined, the approval package is sent to an agent acting in the *approval role* for final approval before execution. The *approval role* validates the *plan* and checklist and may perform additional checks before granting permission to proceed. For example, coordination with other station activities and resources may be managed at this level to obtain final permission to proceed. Finally, authorization for the approval package is sent back to the agent acting in the *user role*.

At this point, the system has a fully authorized *task plan* that contains one or more *atomic tasks* ready for execution. The agent in the *user role* will send the agent in the *task execution role* the authorized *task* corresponding to the first step in the plan. In addition, the *taskmeasurement* and *assessment* information for this *atomic task* will be sent to the agent in the *evaluation role* for future use in determining the success/failure of the executed *task*. During execution, the agent in the *task execution role* examines zero or more preconditions that must be satisfied before *task* execution is allowed to commence. Once satisfied, a *task attempt* is conducted.

Status may be reported to the agent in the *evaluation role* during and/or after the *task attempt*. If the *atomic task* was successful, the cycle will continue with the next *atomic task* being executed. Once the entire *plan* has been executed, relevant *state variables* and *parameters* are sent to the agent in the *evaluation role* for use in *measurement* and *assessment*. A final *outcome* is computed, and an OWLS::Result structure is completed and returned to the agent in the *user role*.

A.3 Specific task workflow

The hierarchical system is designed to accomplish the module installation. The level 1 system receives a *task* of assembling a particular module. During its *task execution*, it will generate a list of *tasks* to be dispatched to the level 2 system and will dispatch these *tasks*. The level 2 system receives these *tasks* and further decomposes each *task* for system execution as a behavior tree. Details are presented in A.3.1 and A.3.2.

A.3.1 Level 1: Assemble

The overall system is designed to be flexible in its operation and work with any electrical module. Therefore, no preset *plan* for how to install this module is necessary. The initial job of the level 1 system is to create an assembly *plan* based on the current state and the requirements of the electrical module. The level 1 system will then dispatch this multi-step *plan* to level 2, thus causing the assembly to take place. Various roles for an agent are shown in the sequence diagram of Figure A.2. An agent, acting in the *specification role*, specifies which module to install and what a correct module installation requires. This is sent to a *user role* agent in the form of a task specification package. This package includes the specification of the desired outcome (the module to be installed), planning domain information, and instance information such as the robot or other resources that are available to be utilized.

Upon receipt of this package, the agent in the *user role* begins a three-step task planning process, as follows:

a) Generate a PDDL problem file which represents the current state and available resources.

b) Solve the problem domain utilizing this generated file with a PDDL solver. The domain for the problem is included in the task specification package. The domain contains parameterized actions that are understood by the level 2 system along with preconditions and anticipated effects of each action. A sample action is shown in Figure A.3. The planning system creates an ordered set of parameterized *actions* that are to be accomplished by the level 2 system. One such set of *actions* is shown in Table A.1.

c) In the last stage of the planning process, the system takes these PDDL formatted commands and composes task templates to be sent to the level 2 system. The templates specify each *task* that needs to be accomplished, but not how to accomplish that *task*. The planning and execution of the *task* is the responsibility of the level 2 system.

```
(:durative-action·insert¶
····:parameters·(?robot·-·robot·?fixturable·-·fixturable·?fixture·-·
fixture·?grasp·-·grasp·?fixture_type·-·fixture_type)¶
····:duration·(·=·?duration·1)¶
····:condition(and¶
········(overall(robot_init·?robot))¶
········(at·start(robot_holds_fixturable·?robot·?fixturable·?grasp))¶
········(at·start(fixture_is_type·?fixture·?fixture_type))¶
········(at·start(fixturable_is_type·?fixturable·?fixture_type))¶
········(at·start(fixture_empty·?fixture))¶
····)¶
····:effect(and¶
········(at·end(in_fixture·?fixturable·?fixture))¶
········(at·end(robot_holds_none·?robot))¶
········(at·end(not·(fixture_empty·?fixture)))¶
········(at·end(not·(robot_holds_fixturable·?robot·?fixturable·?grasp)))¶
····)¶
)¶
```

Figure A.3—Insert action from domain problem with simplified preconditions and effects

Table A.1—Sequential list of subtasks necessary to perform the level 1 task of module installation

Command	Robot	Parameters
init_robot	arm1	
attach_tool	arm1	finger_gripper tool_holder2
extract	arm1	module1 modulesocket1 direct_grasp finger_gripper
insert	arm1	module1 modulesocket2 direct_grasp module_type
detach_tool	arm1	finger_gripper tool_holder1
attach_tool	arm1	nut_driver tool_holder3
extract	arm1	nut1 stud1 nutdriver_grasp nut_driver nut_type
insert	arm1	nut1 modulesocket2_stud1 nutdriver_grasp nut_type
extract	arm1	nut2 stud2 nutdriver_grasp nut_driver nut_type

Table continues

Table A.1—Sequential list of subtasks necessary to perform the level 1 task of module installation *(continued)*

Command	Robot	Parameters
insert	arm1	nut2 modulesocket2_stud2 nutdriver_grasp nut_type
detach_tool	arm1	nut_driver tool_holder2
attach_tool	arm1	finger_gripper tool_holder1
extract	arm1	vga1 vgasocket1 direct_grasp finger_gripper vga_type
insert	arm1	vga1 module1_vga direct_grasp vga_type
extract	arm1	bnc1 bncsocket1 direct_grasp finger_gripper bnc_type
insert	arm1	bnc1 module1_bnc direct_grasp bnc_type
extract	arm1	c7p1 c7psocket1 direct_grasp finger_gripper c7p_type
insert	arm1	c7p1 module1_c7p direct_grasp c7p_type

At this point, as shown in Figure A.2, task authorization can be performed to examine and approve the task templates.

A.3.2 Level 1: Task accomplishment

The system now moves into a loop that includes task initialization, task execution, and task evaluation. As shown in Figure A.4, for each task template task initialization is performed by validating that the template's preconditions are valid. Information on expected effects of the *task* is also transmitted to the agent in the evaluation role. Assuming that the preconditions are met, the *task* is dispatched to the level 2 system for execution as a *task attempt*. At the conclusion of the *task attempt*, returned status from the level 2 system along with relevant state information is sent to the *evaluation role* to determine the success/failure of the *task attempt*. Success shall lead to the execution of the next template, while failure shall result in a system exit.

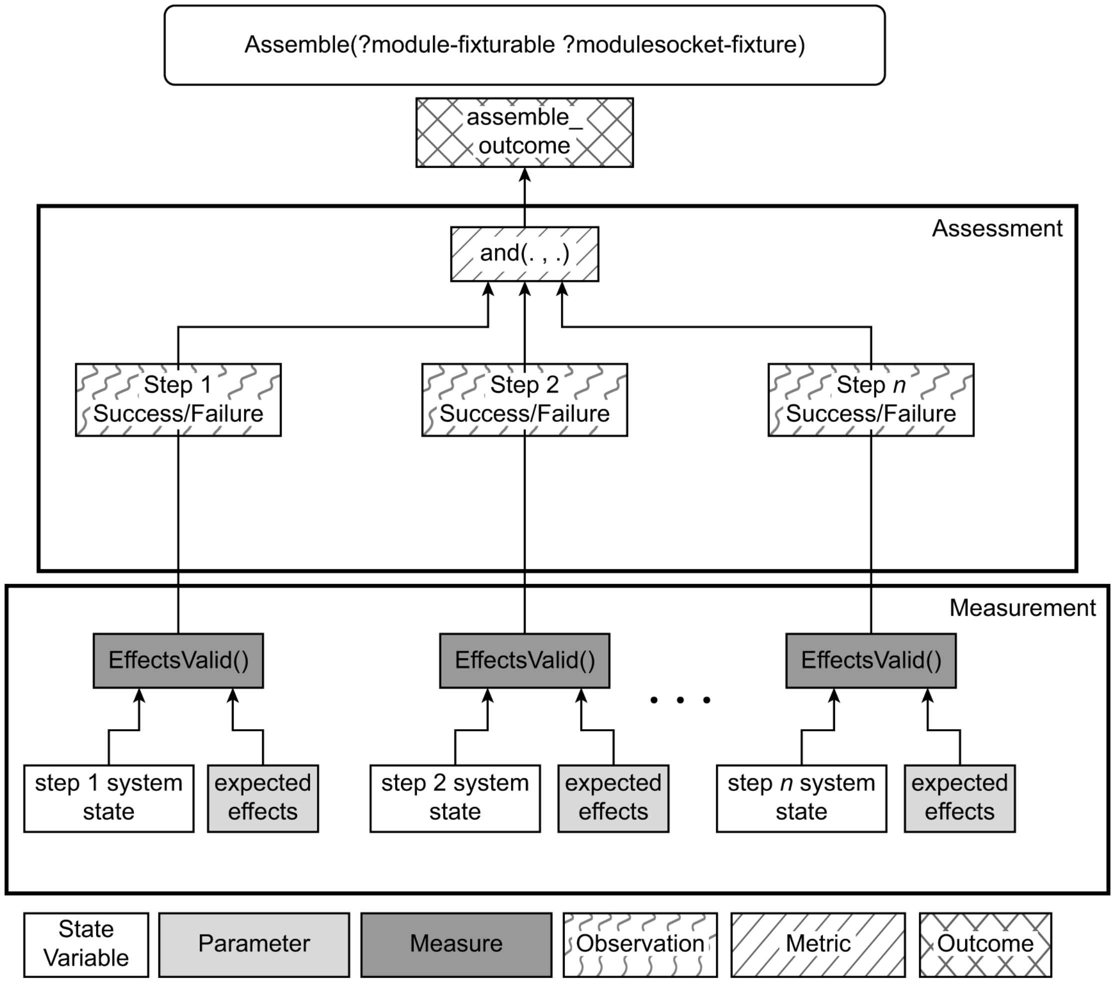

Figure A.4—Outcome structure for level 1 system

A.4 Level 1 schema

The information in Figure A.4 shall be formatted in a computer readable fashion to be utilized as a task template. For this reason, the schema depicted in Figure A.5 has been developed. This schema contains several sections of information that are described below. The exact format of the representation is beyond the scope of this example.

— Composite Task Information: This is information that is relevant to the *composite task*.

  — State_Variables: This section of the schema represents system state information that is necessary to ascertain the proper functioning of the system. The *state variables* provide locations to store the result of the level 1 task execution.

  — Parameters: This section contains information that is necessary to turn the task template into a plan. In the case of the level 1 system, this is the module that is to be assembled and any information that will be required by level 2 systems.

  — Preconditions: This section contains information on preconditions that must be satisfied before the level 1 task is begun. For level 1, the only precondition for the system is that the module is not powered before assembly is begun.

- Atomic Task Information: The level 1 system will fill in task templates for *atomic tasks* that are sequenced to subordinate processes. This section contains information necessary to complete these templates.
    - State_Variables: An array of *state variables* (one entry per *atomic task*) that will contain information necessary to ascertain the proper functioning of the *atomic task*. Note that this is different than having a success/failure returned from the *atomic task*. An *atomic task* may fail but still accomplish its *task*, just as it may succeed yet fail to accomplish its *task*.
    - Parameters: An array of *parameters* (one entry per *atomic task*). This is a mapping of *composite task parameters* to the parameterization of the atomic tasking.
    - Preconditions: An array of preconditions (one entry per *atomic task*). This section contains information on preconditions that must be satisfied before the level 2 *task* is begun. This information is contained in the level 2 task templates. See Figure A.3 for an example.
- Control_Construct: The *control construct* is utilized to specify that the level 1 *task* is a *composite task*. An OWL construct of sequence is utilized to specify the *atomic tasks* that make up this *composite task*. This is in essence the computed plan that must be filled in.
- Outcome: This specifies the *outcome* shown in Figure A.4. The exact language to specify this is beyond the scope of this example. One potential language would be PDDL.
- Exits: *Exits* are part of the OWLS:Result structure. They are represented by a condition code and the resulting *state variables* that are set and effects that occur due to the *exit* being triggered.

CompositeTask : Assemble

- state_variables:
  - assemble_outcome
- parameters:
  - expected effects
- precondition:
  - module_power_off
- AtomicTask state_variables[]
- AtomicTask parameters[]
- AtomicTask preconditions[]
- control_construct:
  - owl-s::sequence:
    - step2(AtomicTask state_variables[0], AtomicTask preconditions[0], AtomicTask parameters[0])
    - step2(AtomicTask state_variables[1], AtomicTask preconditions[1], AtomicTask parameters[1])
    - ...
    - step2(AtomicTask state_variables[*n*], AtomicTask preconditions[*n*], parameters[*n*])
- outcome:
  - assemble_outcome - Pointer to some external source or language construct that composes the outcome.
  - Items that must be available for outcome computation
    - Measures needed for computing the outcome
      - EffectsValid()
    - Metrics needed for computing the outcome
      - and()
    - State variables needed for computing the outcome
      - Provided above
    - Parameters needed for computing the outcome
      - Provided above
- exits:
  - nominal:
    - inCondition: assemble_outcome == success
    - resultVar: assemble_success
    - output: assemble_success
    - effect:
      - Assembly_Complete(module1)
  - faulted:
    - inCondition: assemble_outcome == failure
    - resultVar: assemble_failure
    - output: assemble_failure
    - effect: Notify_User()

Figure A.5—Discussion of the level 1 schema for tasking

A.4.1 Level 2: Assembly planning

The level 2 system accepts several different task templates that correspond to the various commands defined in the original domain file. Each command also corresponds to a behavior tree that will be executed as part of the *task*.

A.5 Level 2 outcomes

Figure A.6 shows a sample *outcome* for the insert *task* that is sent from the level 1 system to the level 2 system. In this case, the inserted connector shall be within a distance tolerance of the mating connector and shall also have passed a test that utilizes the force/torque sensor on the arm to verify that the connector is properly seated. The level 2 system receives information that is necessary to instantiate a behavior tree as its planning process. This behavior tree is then executed and the resulting *measures* evaluated to determine task success.

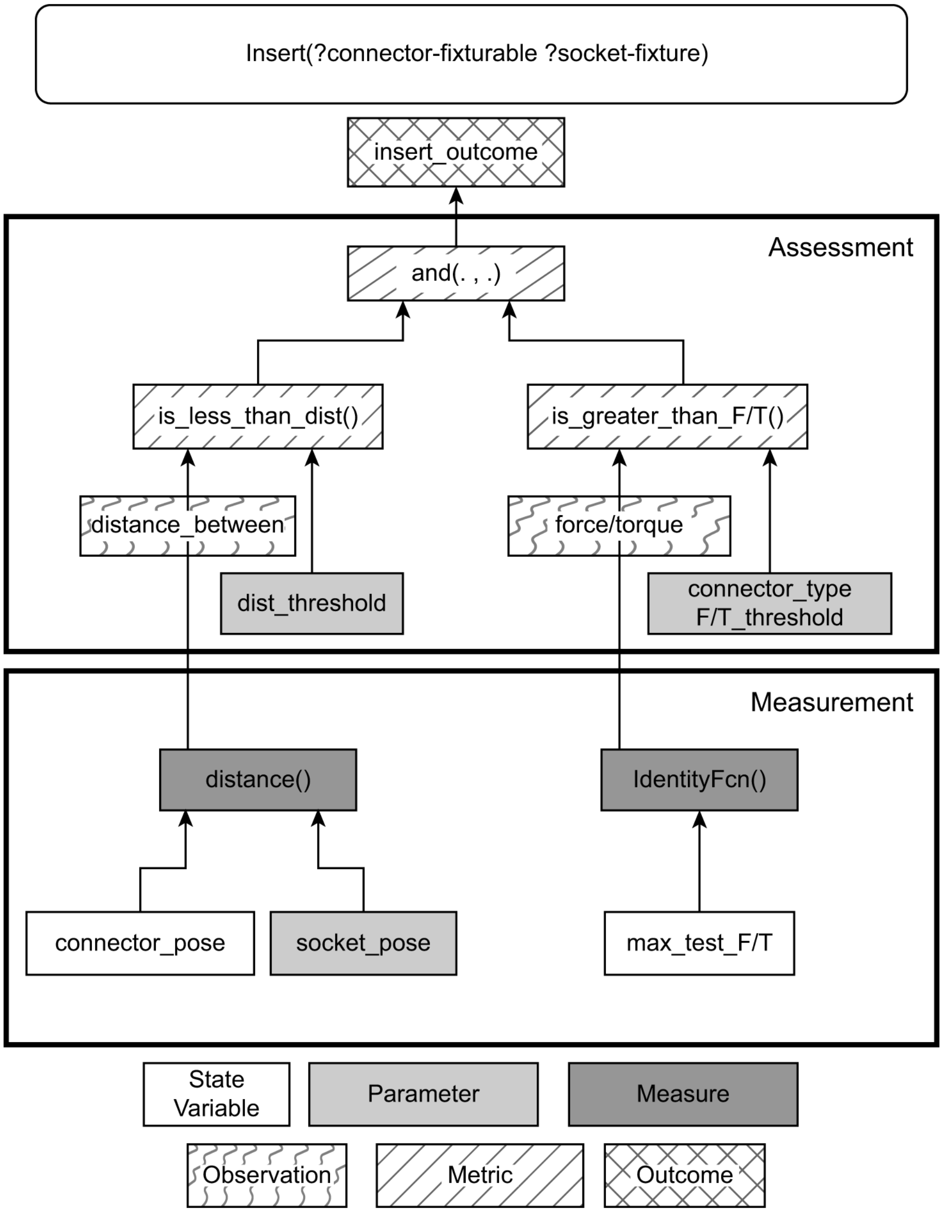

Figure A.6—Sample task template sent to level 2 system

There will be a different template for each command that is shown in Table A.1.

A.6 Level 2 schema

The information from Figure A.7 shall be formatted to in a computer readable fashion in order to be utilized as a task template. The schema is similar to the one developed for the level 1 task framework.

AtomicTask : Insert

- state_variables:
  - connector_pose
  - max_test_ft
- parameters:
  - robbie - robot
  - HDMI_1 - fixturable
  - HDMI_Socket_1 - fixture
- precondition:
  - Robot_Holds(robbie, HDMI_1)
  - Fixture_Type(HDMI_Socket_1, HDMI)
  - Fixtureable_Type(HDMI_1, HDMI)
  - Fixture_Empty(HDMI_Socket_1)
- outcome:
  - insert_outcome - Pointer to some external source or language construct that composes the outcome as shown in the figure "Goal Outcome".
  - Items that must be available for outcome computation
    - Measures needed for computing the outcome
      - identityFcn()
      - distance()
    - Metrics needed for computing the outcome
      - and()
      - is_less_than_dist()
      - is_greater_than_F/T()
    - State variables needed for computing the outcome
      - List is already provided above
    - Parameters needed for computing the outcome
      - List is already provided above
- exits:
  - nominal:
    - inCondition: insert_outcome == success
    - resultVar: insert_success
    - output: insert_success
    - effect:
      - ~Fixture_Empty(HDMI_Socket_1)
      - ~Robot_Holds(robbie, HDMI_1)
      - Fixture_Holds(HDMI_Socket_1, HDMI_1)
  - faulted:
    - inCondition: insert_outcome == failure
    - resultVar: insert_failure
    - output: insert_failure
    - effect: World_Model_Update_Needed()

Figure A.7—Level 2 task schema

IEEE SA
STANDARDS
ASSOCIATION

RAISING THE WORLD'S STANDARDS

Connect with us on:

Facebook: facebook.com/ieeesa
LinkedIn: linkedin.com/groups/1791118
Beyond Standards blog: beyondstandards.ieee.org
YouTube: youtube.com/ieeesa

standards.ieee.org
Phone: +1 732 981 0060

IEEE