An Industrial Robotic Knowledge Representation for Kit Building Applications

Stephen Balakirsky
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
stephen.balakirsky@nist.gov

Zeid Kootbally
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742, USA
zeid.kootbally@nist.gov

Craig Schlonoff
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
craig.schlenoff@nist.gov

Thomas Kramer
Department of Mechanical Engineering
Catholic University of America
Washington, DC 20064, USA
thomas.kramer@nist.gov

Satyandra K. Gupta
Maryland Robotics Center
University of Maryland
College Park, MD 20742, USA
skgupta@umd.edu

***Abstract*— The IEEE RAS Ontologies for Robotics and Automation Working Group is dedicated to developing a methodology for knowledge representation and reasoning in robotics and automation. As part of this working group, the Industrial Robots sub-group is tasked with studying industrial applications of the ontology. One of the first areas of interest for this subgroup is the area of kit building or kitting which is a process that brings parts together in a kit and then moves the kit to the assembly area where the parts are used in the final assembly. Kitting itself may be viewed as a specialization of the general bin-picking problem. This paper examines the knowledge representation that has been developed for the kitting problem and presents our real-time implementation of the knowledge representation along with a discussion of the trade-offs involved in its design.**

I. INTRODUCTION

Kitting is the process in which several different, but related items are placed into a container and supplied together as a single unit. In industrial assembly of manufactured products, kitting is often performed prior to final assembly. Manufacturers utilize kitting due to its ability to provide cost savings [3] including saving manufacturing or assembly space [13], reducing assembly workers walking and searching times [15], and increasing line flexibility [2] and balance [10].

Several different techniques are used to create kits. A kitting operation where a kit box is stationary until filled at a single kitting workstation is refereed to as *batch kitting*. In *zone kitting*, the kit moves while being filled and will pass through one or more zones before it is completed. This paper focuses on batch kitting processes.

In batch kitting, the kit's component parts may be staged in containers positioned in the workstation or may arrive on a conveyor. Component parts may be fixtured, for example placed in compartments on trays, or may be in random orientations, for example placed in a large bin. In addition to the kit's component parts, the workstation usually contains a storage area for empty kit boxes as well as completed kits.

Kitting, has not yet been automated in many industries where automation may be feasible. Consequently, the cost of building kits is higher than it could be. We are addressing this problem by building models of the knowledge that will be required to operate an automated kitting workstation in an agile manufacturing environment. This workstation must be able to cope with variations in kit contents, kit layout, and component supply. We also plan to develop a simulated kitting workstation for model validation. Our models include representations for non-executable information about the workstation such as information about a robot, parts, kit designs, grippers, etc., models of executable information such as actions, preconditions, and effects, and models of the process plan necessary for kit construction. A discussion of the functional requirements for the process plan may be found in [1]. For our automated kitting workstation, we assume that a robot performs a series of pick-and-place operations in order to construct the kit. These operations include:

1) Pick empty kit and place on work table.
2) Pick multiple component parts and place in kit.
3) Pick completed kit and place in full kit storage area.

Each of these actions may be a compound action that includes other actions such as end-of-arm tool changes, path planning, and obstacle avoidance.

It should be noted that multiple kits may be built simultaneously. Finished kits are moved to the assembly floor where components are picked from the kit for use in the assembly procedure. The kits are normally designed to facilitate component picking in the correct sequence for assembly. Component orientation may be constrained by the kit design in order to ease the pick-to-assembly process. Empty kits are returned to the kit building area for reuse.

Although the knowledge requirements described in the

previous paragraph have been identified for the kitting domain, they are clearly applicable to many types of industrial robot applications (and likely to robot applications in other fields). As such, we expect that these knowledge requirements will serve as the basis for the industrial robot ontology being developed in the IEEE RAS Ontologies for Robotics and Automation Working Group [11] (henceforth referred to as the IEEE WG). Throughout the process of developing the kitting ontology, the group will constantly look at the applicability of the requirements outside of kitting and move the pertinent knowledge "up" the ontology (whether in the portion that models the kitting sub-domain, the industrial robot domain, or the upper ontology), as appropriate.

In keeping with our philosophy of producing standards in conjunction with the IEEE WG, we wish the models being developed by this effort to be as widely applicable as possible. To support this desire,we have created a layered model abstraction where users may adopt as many or few of the layers of the abstraction as make sense for their specific application. The architecture shown in Figure 1, though developed for the implementation of the kitting ontology, can be equally applicable to the implementation of any type of formal manufacturing knowledge representation. Said in a different way, the implementor can plug in a knowledge representation for a different domain and the architecture would still be valid. In a similar manner, different planning language abstractions could be utilized in the middle-layer of the abstraction and different planning/execution systems could be utilized in the top-layer of the abstraction.

Specifics on the overall architecture may be found in Section II. The lowest layer of the abstraction is based on the Web Ontology Language (OWL) [18] and is discussed further in Section III. The next layer of the abstraction is modeled in the Planning Domain Definition Language (PDDL) [8]. More information on this layer may be found in Section IV. In order to validate our models, we intend to utilize simulation in conjunction with domain independent planning systems and the Robot Operating System (ROS) [1] control environment [6]. More information on this may be found in Section V. An example of the various knowledge representations and the flow from one to the next is presented in Section VI. Finally, conclusions and future work may be found in Section VII.

II. Architecture Description

The main focus of this work is on the development of knowledge models that allow a kitting workstation to construct kits in an agile manufacturing environment. However, in order to validate these knowledge models, we felt that it was important to be able to utilize the models to construct kit building plans, and then to execute these plans in dynamic virtual and real environments. Due to the advent of open source robotic operating systems such as ROS [6] and simulation packages such as USARSim [4] we do not need to design these systems ourselves. However, our architecture must be designed to represent the required knowledge base in several different abstractions that are likely to be required by these systems as the knowledge flows from domain and process specification, to plan generation, to plan execution. As shown in Figure 1, the abstraction is decomposed into four distinct layers of *Domain Specific Information*, *Ontology*, *Planning Language*, and *Robot Language* that correspond to these knowledge requirements. Implementors of the abstraction are free to connect to the knowledge interface at the layer that makes sense for their particular application. For our simulated kitting workstation, we intend to fully design the Domain Specific Information and the Ontology and then utilize open source tools that will automatically generate the remaining layers and provide a workstation simulation.

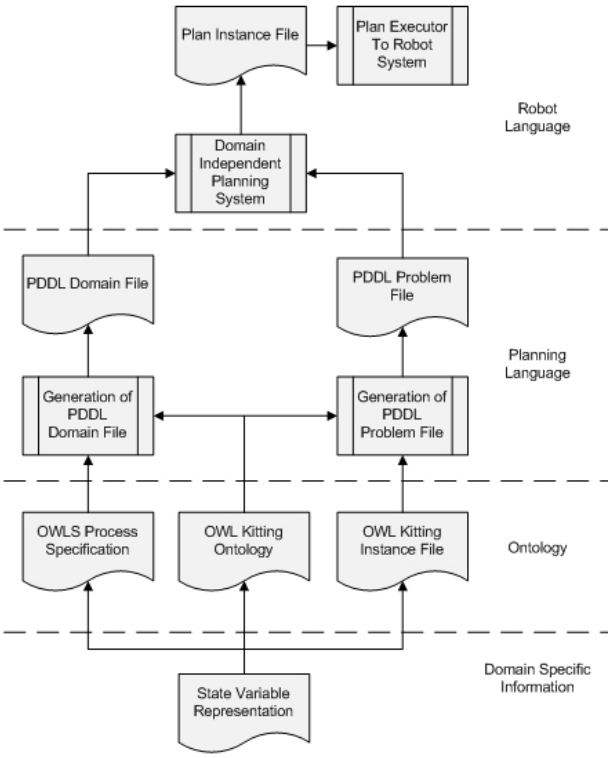

Fig. 1. Kitting data flow abstraction.

A. Domain Specific Information

The foundation for the knowledge representation is domain specific information that is produced by an expert in the particular field of study. This includes information on items ranging from what actions and attributes are relevant, to what are the necessary conditions for an action to occur and what are the likely results of that action. We have chosen to encode this basic information in a formalism know as a state-variable representation (SVR) [14]. This information will then flow up the abstraction and be transformed into the ontology, planning language, and robot language. In a SVR, each state is represented by a tuple of values of n

[1]Certain commercial software and tools are identified in this paper in order ot explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

state variables $\{x_1, \ldots, x_n\}$, and each action is represented by a partial function that maps this tuple into some other tuple of values of the n state variables.

B. Constant Symbols for the Kitting Domain

For the kitting domain, there is a finite set of constant symbols that must be represented in the system. In the SVR, constant symbols are partitioned into disjoint classes corresponding to the objects of the domain. The finite set of all constant symbols in the kitting domain is partitioned into the following sets of constant symbols:

- A set of *Robots* $\{r_1, r_2, \ldots\}$: A *Robot* in the kitting workstation is a robotic arm that can move objects in order to build $KitInstances$.
- A set of $KitTrays$ $\{kt_1, kt_2, \ldots\}$: A $KitTray$ is a tray with which a $KitInstance$ may be built. A $KitTray$ can hold parts in known positions.
- A set of $KitInstances$ $\{kins_1, kins_2, \ldots\}$: A $KitInstance$ is built when $Parts$ are placed in a $KitTray$. A $KitInstance$ consists of a $KitTray$ and, possibly, some $Parts$. A $KitInstance$ is empty when it does not contain any $Part$ and finished when it contains all the $Parts$ that constitute a kit.
- A set of $PartTrays$ $\{pt_1, pt_2, \ldots\}$: $Parts$ arrive at the workstation in $PartTrays$. Each part is at a known position in the $PartTray$. Each $PartTray$ contains one type of $Part$.
- A set of $Parts$ $\{p_1, p_2, \ldots\}$: $Parts$ are delivered to the workstation in $PartTrays$ and are placed in $KitTrays$ to make $KitInstances$.
- A set of $EndEffectors$ $\{eeff_1, eeff_2, \ldots\}$: $EndEffectors$ are used in a kitting workstation to manipulate $Parts$, $PartTrays$, $KitTrays$, and $KitInstances$.
- A set of $EndEffectorHolders$ $\{eeffholder_1, eeffholder_2, \ldots\}$: An $EndEffectorHolder$ holds one type of $EndEffector$.
- A symbol $EndEffectorChangingStation$ – $chstation$: An $EndEffectorChangingStation$ is made up of $EndEffectorHolders$.
- A set of $LargeBoxWithKits$ $\{lbwk_1, lbwk_2, \ldots\}$: A $LargeBoxWithKits$ contains only finished $KitInstances$.
- A set of $LargeBoxWithEmptyKitTrays$ $\{lbwekt_1, lbwekt_2, \ldots\}$: A $LargeBoxWithEmptyKitTrays$ is a box that contains only empty $KitTrays$.
- A symbol $WorkTable$ – $wtable$: A $WorkTable$ is an area in the kitting workstation where $KitTrays$ are placed to build $KitInstances$.

C. State Variable Symbols for the Kitting Domain

the next sentence is confusing...

State variable symbols are functions from the set of states and zero or more sets of constant variables into a set of constant variables [14]. Using state variable symbols reduces the possibility of inconsistent states and generates a smaller state space. The following state variable symbols are used in the kitting domain:

- *geffloc*: EndEffectors $\times$ S $\rightarrow$ Robots $\cup$ EndEffectorHolders: designates the location of an EndEffector in the workstation. The EndEffector can be placed in a EndEffectorHolder or attached to the Robots.
- *rgrip*: Robots $\times S \rightarrow$ EndEffectors $\cup \{nil\}$: designates the EndEffector that is attached to a Robots or nil if no EndEffector is attached.
- *topworktable*: WorkTable $\times$ S $\rightarrow$ KitInstancess $\cup$ $\{nil\}$: designates the object placed on the WorkTable, it can be either a KitInstnce or nothing (nil).
- *kinsloc*: KitInstances $\times$ S $\rightarrow$ LargeBoxWithKits $\cup$ WorkTable $\cup$ Robots: designates the different possible locations of a KitInstance in the workstation. The KitInstance can be in a LargeBoxWithKits, on the WorkTable, or being held by the Robot.
- *ktloc*: KitTrays $\times$ S $\rightarrow$ LargeBoxWithEmptyKitTrays $\cup$ Robots: designates the different possible locations of a KitTray in the workstation. The KitTray can either be in a LargeBoxWithEmptyKitTrays or being held by the Robot.
- *ploc*: Parts $\times$ S $\rightarrow$ PartTrays $\cup$ KitInstances $\cup$ Robots: designates the different possible locations of a Part in the workstation. The Part can be in a PartTray, in a KitInstance, or being held by the Robot.
- *rhold*: Robots $\times$ S $\rightarrow$ KitTrays $\cup$ KitInstances $\cup$ Parts $\cup \{nil\}$: designates the object being held by the Robots. It can be a KitTray, a KitInstance, Part, or nothing (nil). It is assumed that the robot is already equipped with the appropriate EndEffector.
- *islbwkfull*: LargeBoxWithKits $\times$ S $\rightarrow$ $\{0\} \cup \{1\}$: designates if a LargeBoxWithKits is full (1) or not (0).
- *islbwektempty*: LargeBoxWithEmptyKitTrays $\times$ S $\rightarrow$ $\{0\} \cup \{1\}$: designates if a LargeBoxWithEmptyKitTrays is empty (1) or not (0).
- *isparttrayempty*: PartTrays $\times S \rightarrow \{0\} \cup \{1\}$: designates if a PartTray is empty (1) or not (0).
- *eefftype*: EndEffectors $\times$ S $\rightarrow$ KitTrays $\cup$ Parts: designates the type of object the EndEffector can hold. For the kitting domain used in this paper, an EndEffector can hold two types of object: KitTrays and Parts.

D. Planning Operators and Actions for the Kitting Domain

A planning operator [14] is a triple *o=(name(o), precond(o), effects(o))* where:

- name(o) is a syntactic expression of the form $n(u_1, \ldots, u_k)$, where n is a symbol called an operator symbol, $u_1, \ldots, u_k$ are all of the object variable symbols that appear anywhere in o, and n is unique (i.e., no two operators can have the same operator symbol).
- precond(o) is a set of expressions on state variables and relations.

- effects(o) is a set of assignments of values to state variables of the form $x(t_1, \ldots, t_k) \leftarrow t_{k+1}$, where each t_i is a term in the appropriate range.

The kitting domain is composed of eight operators. Only the name of the operators along with a description are mentioned in the rest of this section. Section VI shows an example of a detailed operator (name, precond, and effects).

1) *take-kt* (r, kt, lbwekt, eeff): The Robot r is equipped with the EndEffector eeff to pick up the KitTray kt from the LargeBoxWithEmptyKitTrays lbwekt.
2) *put-kt* (r, kt, wtable): The Robot r puts down the KitTray kt on the WorkTable wtable.
3) *take-kins* (r, kins, wtable, eeff): The Robot r picks up the KitInstance kins from the WorkTable wtable.
4) *put-kins* (r, kins, lbwk): The Robot r puts down the KitInstance kins in the LargeBoxWithKits lbwk.
5) *take-p* (r, p, pt, eeff): The Robot r uses the EndEffector eeff to pick up the Part p from the PartTray pt.
6) *put-p* (r, p, kins): The Robot r puts down the Part p in the KitInstance kins.
7) *attach-eeff* (r, eeff, eeffholder): The Robot r attaches the EndEffector eeff from the EndEffectorHolder eeffholder.
8) *remove-eeff* (r, eeff, eeffholder): The Robot r removes the EndEffector eeff and puts it in the EndEffectorHolder eeffholder.

E. Ontology

"Ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences" [19].

Knowledge models may take many forms ranging from informal natural language, to XML schemas, to ontologies. Since this work is being directed at the IEEE RAS Ontologies for Robotics and Automation Working Group, it is appropriate that our knowledge representation be encoded in an ontology at the lowest layer of our abstraction. The knowledge contained in this layer is sufficient for a planning system to understand the domain and construct a plan for creating the desired kit from the given resources.

As shown in Figure 1 the information on our ontology is divided into three files and consists of a representation of the process specification, the kitting ontology, and the instance file. The process specification file contains descriptions of the individual actions and sequences necessary to construct a kit, e.g. gripping a component from a tray. The kitting ontology file contains the concepts related to the specific items that compose the kitting domain, e.g. a full description of each component including items such as the component's weight, dimensions, and grip points. The instance file contains specific information on *this* particular kitting problem and configuration, e.g. component tray 1 contains 4 components of type 'A'.

While this file set provides a complete description of the problem domain and environment, most planning systems cannot directly ingest information from an ontology. Therefore, the second layer of the data abstraction known as the PDDL Conformance layer was created.

F. PDDL Conformance

By placing our knowledge in a domain independent representation, we enable the use of an entire family of open source planning systems. In particular, we have chosen to transform the information stored in the lower layer of our abstraction into the Planning Domain Definition Language (PDDL) [8]. This provides our second layer of standardized output that other systems may connect to. As shown in Figure 1, this information consists of two files. The first is a PDDL domain file that specifies all of the planning axioms and object classes and is a summary of the process specification and ontology files. The second is a PDDL problem file that specifies the system's initial state and goal state.

G. Robot Plan Conformance

The PDDL output may then be fed into a number of open source planning systems that will produce a plan instance file that is based on our original vocabulary. This file contains a time sequenced series of actions that must be carried out in order to create a transition from our initial system state to the goal system state. Up to this point, the entire architecture is independent of the kitting domain or any specific hardware configurations and is able to solve problems for many types of industrial robot applications. The top layer of our architecture ties the specific commands to the kitting workstation and is dependent on the workstation's resident hardware. In order to maintain as much hardware independence as possible, we have chosen to use the ROS environment for communicating with our simulation. Therefore, the PDDL commands will be translated into appropriate commands that will be sent into various ROS processes.

III. OWL Model of Manufacturing Process

For the development of the knowledge representation, the industrial robots sub-group has decided to use OWL [18] as the knowledge representation language. OWL is a family of knowledge representation languages for authoring ontologies and is endorsed by the World Wide Web Consortium (W3C). It is characterized by formal semantics and RDF/XML-based serialization for the Semantic Web. OWL was chosen by the group because of its popularity among the ontology development community, its endorsement by the W3C, as well as the number of tools and reasoning engines that are available. OWL was also selected as the representation language that will be used in the overall IEEE WG efforts.

In addition to OWL, the industrial robots subgroup will also be using OWL-S [12] to represent the processes and actions that the robot will perform. OWL-S is an ontology built on top of OWL by the DARPA Agent Markup Language (DAML) program [5] for describing Semantic Web Services. However, many of the constructs that are used to describe services are equally applicable to describing robot actions. For example, concepts such as preconditions, results, inputs,

outputs, effects, and participants are generic enough to be applied to just about any type of process specification.

To build the ontology, the group has taken a very systematic approach of identifying and modeling the concepts. Because the industrial robot field is so broad, the group decided to limit its efforts to a single type of operation, namely kitting. A scenario was developed that described, in detail, the types of operations that would be performed in kitting, the sequencing of steps, the parts and machines that were needed, constraints on the process such as pre- and post-conditions, etc. For this scenario, a set of concepts were extracted and defined. These concepts served as the initial requirements for the kitting ontology. The concepts were then modeling in OWL, building off of the definitions and relationships that were identified in the scenario.

As more detailed scenarios are determined and a richer set of concepts are uncovered, the ontology will be partitioned based upon the generality of the concept, with the most generally applicable concepts being "higher" in the ontology so they are available to other domains and the more detailed concepts being "lower" in the ontology because they will likely be very specific to the kitting area. An example of a general concept may be a "robot" while a specific concept may be a "kit box."

Some of the concepts that are represented in the ontology include:

- *GripperEffector* - A *GripperEffector* is an *EndEffector*. A *GripperEffector* holds an object by gripping it with fingers or claws.
- *KitTrays* - A *KitTray* is designed to hold *Parts* with various *StockKeepingUnits* ids (*SKU*) in known positions.
- *KittingWorkStation* - A *KittingWorkstation* contains a *WorkTable*, a *Robot*, an *EndEffectorChangingStation* (where various end-of-arm tooling is stored and attached/removed from the robot), and other fixed obstacles such as a computer. A *KittingWorkstation* has properties that include an *angle unit*, a *length unit*, and a *weight unit*. All angle, length, and weight values related to the workstation must use those units. A *KittingWorkstation* has a list of *SKUs* it knows about. In addition, containers of various sorts enter and leave the workstation. The robot builds kits of parts by executing kitting plans as directed by a kitting plan execution system.
- *Part* - A *Part* has a reference to the identifier of a *SKU* and a *SerialNumber*.
- *PartsBin* - A *PartsBin* holds a number of *Parts* with the same *SKU* in unknown random positions, or fixtured known positions.
- *Robot* - A *Robot* has properties of a *description*, a *robot id*, a *work volume*, an *end effector*, and a *maximum load weight*.
- *VacuumEffector* - A *VacuumEffector* holds an object by putting a cup against the object and applying a vacuum. It contains properties of *maximum lifting force* and the *vacuum cup configuration*.
- *WorkTable* - The top of a *WorkTable* is a flat, rectangular, horizontal surface.

IV. PDDL Model of Manufacturing Process

PDDL is an attempt to standardize planning domain and problem description languages. Rest moved to balakirskyIROS2002.tex for now. Will move some text back...

V. ROS Model of Manufacturing Process

When developing a knowledge representation, it is very difficult to determine if the representation is sufficient for actual use in a dynamic, unstructured environment. One technique that may be utilized that provides repeatable experiments with deterministic amounts of noise and world variance is simulation. As part of our kitting work station implementation, we are developing an open source application that will translate our PDDL command syntax into standard ROS topics for the ROS Arm Navigation Stack known as arm_navigation.

VI. Example of Operation

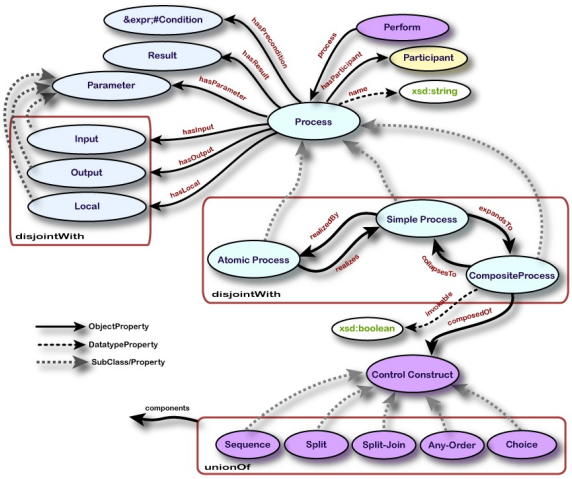

Fig. 2. Selected classes and properties of the profile [16].

A. OWL-S Representation

Once the action is specified in state-variable (S-V) representation, we can use that information to create an OWL-S process. Figure 2 shows a schematic of the OWL-S structure. It is outside of the scope of this paper to describe this figure in detail, but for the sake our example, we are modeling the action as a process and including information about its data inputs and outputs, the preconditions that have to be true for it to be performed, and the result that will be true after the process is executed. In our example, the process is an atomic process because it only involves a single interaction and consists of only one step.

In the S-V representation, the preconditions clearly map to the OWL-S precondition bubble. These can be represented in languages such as KIF [7] (Knowledge Interchange Format), SPARQL [17] (SPARQL Protocol and RDF Query Language), or SWRL [9] (Semantic Web Rule Language). The rules point to classes and instances in the ontology that

model the concepts of kit tray (kt), a set of large boxes with empty kit trays (lbwekt), a robot (r), and a robot gripper effector (eeff). The S-V effects map to the OWL-S results and are also represented in one of the rules languages above. In the case of the *take-kt* action, the result would specify that the location of the kit tray is no longer in a fixed location and is now in the robot gripper effector.

Though not explicitly represented in the S-V representation, data inputs and outputs are an important part of the OWL-S representation and can be inferred from the S-V representation. Specifically, it needs to know which robot is performing the action (r), which kit tray needs to be picked up (kt), which gripper effector is on the robot (eeff), and from which box the robot needs to pick up the kit tray (lbwekt). The output of this action would be a Boolean stating whether the action was completed successfully or not.

VII. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

B. Future Works

REFERENCES

[1] S. Balakirsky, Z. Kootbally, T. Kramer, R. Madhavan, C. Schlenoff, and M. Shneier. Functional Requirements of a Model for Kitting Plans. In *Proceedings of the 2012 Performance Metrics for Intelligent Systems (PerMIS'12)*, 2012.

[2] Y. A. Bozer and L. F. McGinnis. Kitting versus line stocking: A conceptual framework and descriptive model. *International Journal of Production Economics*, 28:1–19, 1992.

[3] O. Carlsson and B. Hensvold. Kitting in a high variation assembly line. Master's thesis, Lule University of Technology, 2008.

[4] S. Carpin, M. Lewis, Jijun Wang, S. Balakirsky, and C. Scrapper. USARSim: A Robot Simulator for Research and Education. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1400 –1405, april 2007.

[5] DARPA. The darpa agent markup language homepage. In *http://www.daml.org*, 2012.

[6] Willow Garage. Robot operating system (ros). In *http://www.willowgarage.com/pages/software/ros-platform*, 2012.

[7] M.R. Genesereth and R.E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. techreport KSL-92-86, Knowledge Systems, AI Laboratory, Stanford, CA, USA, June 1992.

[8] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl–the planning domain definition language. Technical Report CVC TR98-003/DCS TR-1165, Yale, 1998.

[9] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004.

[10] J. Jiao, M. M. Tseng, Q. Ma, and Y. Zou. Generic Bill-of-Materials-and-Operations for High-Variety Production Management. *Concurrent Engineering: Research and Applications*, 8(4):297–321, December 2000.

[11] R. Madhavan, W. Yu, G. Biggs, and C. Schlenoff. Ieee ras standing committee for standards activities: History and status update. *Journal of Advanced Robotics Special Issue on Internationalization*, 2011.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. In *http://www.w3.orag/Submission/OWL-S/*, 2012.

[13] L. Medbo. Assembly work execution and materials kit functinoality in parallel flow assembly systems. *International Journal of Production Economics Journal of Industrial Ergonomics*, 31:263–281, 2003.

[14] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[15] G.F. Schwind. How storage systems keep kits moving. *Material Handling Engineering*, 47(12):43–45, 1992.

[16] W3C Member Submission. Owl-s: Semantic markup for web services. *http://www.w3.org/Submission/OWL-S*, 2004.

[17] W3C. Sparql query language for rdf. In *http://www.w3.org/TR/rdf-sparql-query/*, 2008.

[18] W3C. Owl 2 web ontology language document overview. In *http://www.w3.org/TR/owl-overview/*, 2012.

[19] Wikipedia. Ontology. In *http://en.wikipedia.org/wiki/Ontology*, 2012.