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. An example of the various knowledge representations and the flow from one to the next is presented in Section III. Finally, conclusions and future work may be found in Section IV.

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] [1] 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.

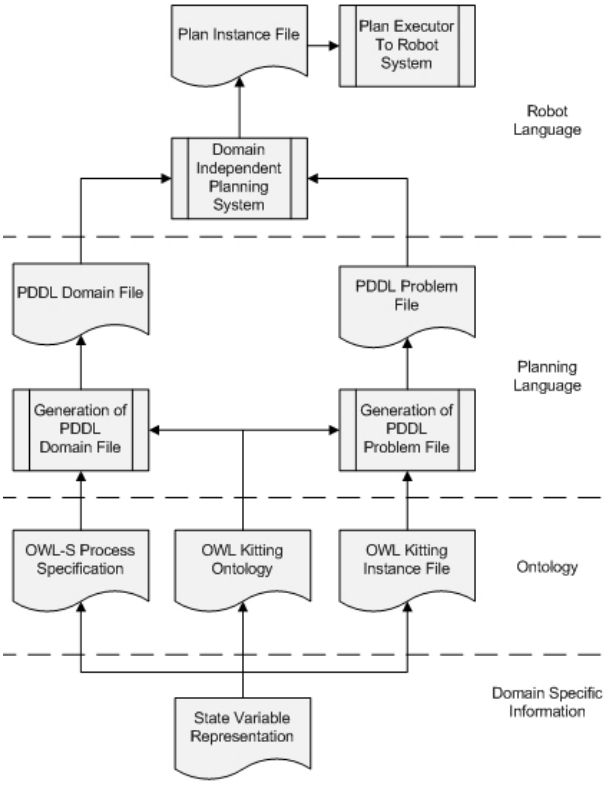

Fig. 1. Kitting data flow abstraction.

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 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.

To build the SVR, 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,

[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.

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 SVR. The concepts were then modeling in our SVR, building off of the definitions and relationships that were identified in the scenario.

A SVR relies on the elements of constant symbols, object variable symbols, state variable symbols, and planning operators. These are defined for the kitting domain in the rest of this section.

1) Constant Symbols: 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 Parts $\{\mathrm{p}_1, \mathrm{p}_2, \ldots\}$: Parts are are the basic items that will be used to fill a kit.
- A set of PartTrays $\{\mathrm{pt}_1, \mathrm{pt}_2, \ldots\}$: 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 KitTrays $\{\mathrm{kt}_1, \mathrm{kt}_2, \ldots\}$: A KitTray can hold Parts in known positions.
- A set of KitInstances $\{\mathrm{kins}_1, \mathrm{kins}_2, \ldots\}$: 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 symbol WorkTable – wtable: A WorkTable is an area in the kitting workstation where KitTrays are placed to build KitInstances.
- A set of LargeBoxWithKits $\{\mathrm{lbwk}_1, \mathrm{lbwk}_2, \ldots\}$: A LargeBoxWithKits contains only finished KitInstances.
- A set of LargeBoxWithEmptyKitTrays $\{\mathrm{lbwekt}_1, \mathrm{lbwekt}_2, \ldots\}$: A LargeBoxWithEmptyKitTrays is a box that contains only empty KitTrays.
- A set of Robots $\{\mathrm{r}_1, \mathrm{r}_2, \ldots\}$: A Robot in the kitting workstation is a robotic arm that can move objects in order to build KitInstances.
- A set of EndEffectors $\{\mathrm{eeff}_1, \mathrm{eeff}_2, \ldots\}$: EndEffectors are used in a kitting workstation to manipulate Parts, PartTrays, KitTrays, and KitInstances. An EndEffector is attached to a Robot.
- A set of EndEffectorHolders $\{\mathrm{eeffholder}_1, \mathrm{eeffholder}_2, \ldots\}$: An EndEffectorHolder is a storage unit that holds one type of EndEffector.
- A symbol EndEffectorChangingStation – $\mathrm{chstation}$: An EndEffectorChangingStation is made up of EndEffectorHolders.

2) Object Variable Symbols: Object variable symbols are typed variables which range over a class or the union of classes of constants. Examples of object variable symbols are $\mathrm{r} \in \mathrm{Robots}$, $\mathrm{kt} \in \mathrm{KitTrays}$, etc.

3) State Variable Symbols: A state variable symbol x: $\mathrm{A}_1 \times \ldots \times \mathrm{A}_\mathrm{i} \times S \rightarrow \mathrm{B}_1 \cup \ldots \cup \mathrm{B}_\mathrm{j}$ $(i, j \geq 1)$ is a function from the set of states (S) and at least one set of constant variable symbols ($\mathrm{A}_1 \times \ldots \times \mathrm{A}_\mathrm{i}$) into a set of constant variable symbols ($\mathrm{B}_1 \cup \ldots \cup \mathrm{B}_\mathrm{j}$). 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:

- *eEffLoc*: 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 $\times S \rightarrow$ KitInstances $\cup \{nil\}$: 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.

4) Planning Operators and Actions: 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 operators' names and descriptions are mentioned in the rest of this section. Section III describes a more detailed operator through an example.

1) *take-kt* (r, kt, lbwekt, eeff): The Robot r equipped with the EndEffector eeff picks 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 equipped with the EndEffector eeff 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.

An action *a* can be obtained by substituting the object variable symbols that appear anywhere in *o* with constant symbols. For instance, the operator *take-p*(r, p, pt, eeff) in the kitting domain can be translated into the action *take-p*(r_1, p_1, pt_1, eeff_2) where r_1, p_1, pt_1, and eeff_2 are constant symbols in the classes Robots, Parts, PartTrays, and EndEffectors, respectively.

B. 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" [18].

Knowledge models may take many forms ranging from informal natural language, to XML schemas, to ontologies. For the development of the knowledge representation, the industrial robots sub-group has decided to use the Web Ontology Language (OWL) [17] 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 Resource Description Framework/eXtensible Markup Language (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 encoding our SVR. 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.

Since this work is being directed at the IEEE RAS Ontologies for Robotics and Automation Working Group, it is appropriate that our domain specific knowledge be encoded in an OWL ontology at the first domain independent layer of our abstraction. The knowledge contained in this layer is derived from our SVR and may also contain information that has been collected over other domains. The knowledge is sufficient for a planning system to understand the specific problem domain and construct a plan for creating the desired kit from the given resources.

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 KitTray.

As shown in Figure 1, the information in 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 is based off of the planning operators from the SVR and 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 is based off of the constant symbols and state variable symbols and contains the concepts related to the specific items that compose the kitting domain, e.g. the hierarchy of what it means to be a $\mathrm{Part_a}$. That a $\mathrm{Part_a}$ is a type of Part, and that all Parts contain properties such as the part's weight, dimensions, and grip points. The instance file is based off of the SVR object variable symbols and contains specific information on *this* particular kitting problem and configuration, e.g. $\mathrm{KitTray kt_1}$ contains 4 Parts of type $\mathrm{Part_a}$.

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 next layer of the data abstraction known as the Planning Language layer was created.

C. Planning Language

The Planning Domain Definition Language (PDDL) [8] is an attempt by the domain independent planning community to formulate a standard language for planning. A community of planning researchers has been producing planning systems that comply with this formalism since the first International Planning Competition held in 1998. This competition series continues today, with the seventh competition being held in 2011. PDDL is constantly adding extensions to the base language in order to represent more expressive problem domains. Our work is based on PDDL Version 3.

By placing our knowledge in a PDDL representation, we enable the use of an entire family of open source planning systems. As shown in Figure 1, each PDDL file-set consists of two files that specify the domain and the problem. The PDDL domain file is composed of four sections that include requirements, types and constants, predicates, and actions. This file may be automatically generated from a combination of information that is contained in the OWL-S process specification file and the OWL Kitting Ontology file.

The requirements section specifies which extensions this problem domain relies on. The planning system can examine this statement to determine if it is capable of solving problems in this domain. In PDDL, all variables that are used in the domain must be typed. Types are defined in the types section. It is also possible to have constants that specify that all problems will share this single value. For example, in the simplest kitting workstation we will have a single $\mathrm{Robot}\ \mathrm{r}_1$. Predicates specify relationships between instances. For example, an instance of a $\mathrm{KitTray}$, kt_1, can have a physical location and contains instances of Parts, Part_a, Part_b, and Part_c. The final section of the PDDL domain file is concerned with actions. An action statement specifies a way that a planner affects the state of the world. The statement includes parameters, preconditions, and effects. The preconditions dictate items that must be initially true for the action to be legal. The effect equation dictates the changes in the world that will occur due to the execution of the action.

The second file of the PDDL file-set is a problem file. The problem file specifies information about the specific instance of the given problem. This file contains the initial conditions and definition of the world (in the init section) and the final state that the world must be brought to (in the goal section). A specific example of the ontology to planning language conversion is provided in Section III.

D. Robot Plan Conformance

Any one of a number of open source planning systems may now be run on the Planning Language layer's knowledge representation. The output of these planners will be a PDDL action sequence 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. It should be noted that the sequence of knowledge transformations that takes place from the ontology through the planning language, to the action sequence 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 abstraction 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. These ROS stacks will then control all aspects of our USARSim virtual workstation.

III. Example of Operation

The purpose of this section is to illustrate the various knowledge representations, depicted in Figure 1, and the flow from one knowledge representation to the next through a specific example.

1) State Variable Representation: In the example, the robot has to build a kit that contains two Parts of type A, one Part of type B and one Part of type C. The kitting process is completed once the kit is placed in the LargeBoxWithKits. Section III-A details the steps that builds the PDDL domain file while section III-B discusses the process that builds the PDDL problem file.

a) Constant Variables: The constant variables available for the example are r_1, kt_1, lbwekt_1, lbwk_1, a symbol wtable, $\mathrm{pt_A}$, $\mathrm{pt_B}$, $\mathrm{pt_C}$, $\mathrm{p_{A-1}}$, $\mathrm{p_{A-2}}$, $\mathrm{p_{B-1}}$, $\mathrm{p_{C-1}}$, eeff_1, eeff_2, $\mathrm{chstation}$, $\mathrm{eeffholder}_1$, and $\mathrm{eeffholder}_2$.

b) Rigid Relation: In the example presented, eeff_1 handles KitTrays whereas eeff_2 handles Parts. These statements can be represented by the rigid state variable symbol *eEffType*:

- *eEffType*(eeff_1)=kt_1
- *eEffType*(eeff_2)=$\{\mathrm{p_{A-1}}, \mathrm{p_{A-2}}, \mathrm{p_{B-1}}, \mathrm{p_{C-1}}\}$

c) Initial State: In the initial state s_0, the Robot r_1 is not equipped with any EndEffector and is not holding anything. The EndEffectors eeff_1 and eeff_2 are placed in the EndEffectorHolders $\mathrm{eeffholder}_1$ and $\mathrm{eeffholder}_2$, respectively. The LargeBoxWithKits lbwk_1 is not full. The LargeBoxWithEmptyKitTrays lbwekt_1 is not empty. The PartTrays $\mathrm{pt_A}$, $\mathrm{pt_B}$, and $\mathrm{pt_C}$ are not empty. The WorkTable wtable is free of any object.

The initial state s_0 can be formulated with the following set of state variable symbols: s_0=$\{$*rHold*(r_1)=nil, *eEffLoc*(eeff_1)=$\mathrm{eeffholder}_1$, *eEffLoc*(eeff_2)=$\mathrm{eeffholder}_2$,*isLBWKFull*(lbwk_1)=0, *isLBWEKTEmpty*(lbwekt_1)=0, *isPartTrayEmpty*($\mathrm{pt_A}$)=0, *isPartTrayEmpty*($\mathrm{pt_B}$)=0, *isPartTrayEmpty*($\mathrm{pt_C}$)=0, *topWorkTable*(wtable)=$nil\}$.

d) Goal State: In the goal state s_g, the KitInstance kins_1 contains Parts $\mathrm{p_{A-1}}$, $\mathrm{p_{A-2}}$, $\mathrm{p_{B-1}}$, and $\mathrm{p_{C-1}}$. The KitInstance kins_1 is placed in the LargeBoxWithKits lbwk_1.

can be formulated with the following set of state variable symbols: s_g =$\{$*pLoc*($\mathrm{p_{A-1}}$)=kins_1, *pLoc*($\mathrm{p_{A-2}}$)=kins_1, , *pLoc*($\mathrm{p_{B-1}}$)=kins_1, *pLoc*($\mathrm{p_{C-1}}$)=kins_1, *kInsLoc*(kins_1)=$\mathrm{lbwk}_1\}$.

e) Action: Eight different operators were introduced in section II-A.4. However, for the sake of this example, the operator *take-kt* (r, kt, lbwekt, eeff) will be used. The preconditions

Substituting the object variable symbols described in the definition of the operator *take-kt* with the constant variables defined in this example, the following action is produced:

- *take-kt* (r_1, kt_1, lbwekt_1, eeff_1)
  - The Robot r_1 equipped with the EndEffector eeff_1 picks up the KitTray kt_1 from the LargeBoxWithEmptyKitTrays lbwekt_1.
  - precond: *kTLoc*$(\mathrm{kt}_1) = \mathrm{lbwekt}_1$, *rHold*$(\mathrm{r}_1) = \mathrm{nil}$, *rGrip*$(\mathrm{r}_1) = \mathrm{eeff}_1$, *isLBWEKTEmpty*$(\mathrm{kt}_1) = 0$, *eEffLoc*$(\mathrm{eeff}_1) = \mathrm{r}_1$
  - effects: *kTLoc*$(\mathrm{kt}_1) \leftarrow \mathrm{r}_1$, *rHold*$(\mathrm{r}_1) \leftarrow \mathrm{kt}_1$

A. Process for the PDDL Domain File

1) OWL-S Representation: Once the action is specified in the SVR, we can use that information to create an OWL-S process. 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. If it were a more complex process that involved multiple sub-actions, it would be modeled by an OWL-S composite process.

In the SVR, the preconditions clearly map to the OWL-S precondition. These can be represented in languages such as KIF [7] (Knowledge Interchange Format), SPARQL [16] (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 SVR 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 SVR, data inputs and outputs are an important part of the OWL-S representation and can be inferred from the SVR. 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.

B. Process for the PDDL Problem File

C. ROS Representation

IV. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the first knowledge model that is part of the IEEE RAS Ontologies for Robotics and Automation, Industrial Robotics subgroup on kitting. This abstraction begins with data that has been created by an expert user as a state variable representation and encoded into an OWL ontology. The knowledge is the automatically transitioned into a Planning Domain Definition Language that may be used by a class of planners to find a solution that will provide for actions that will transition the system for its initial state to a prescribed goal state.

This workstation is still very much a work in progress. The IEEE working group has an active mailing list and meets in person at both the IEEE ICRA and IROS conferences. At present, we are working on connecting the planned action sequence through the ROS control stack to our USARSim virtual world. As the knowledge representation is exercised, it is expected that short comings and missing data will be detected. Our representation will be updated as necessary to cope with these developments. In addition, the representation will be expanded to include metrics that allow us to represent not only the the information necessary for the construction of a kit, but also for the representation of the quality of the finished kit and the kit building process.

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. Sparql query language for rdf. In *http://www.w3.org/TR/rdf-sparql-query/*, 2008.
[17] W3C. Owl 2 web ontology language document overview. In *http://www.w3.org/TR/owl-overview/*, 2012.
[18] Wikipedia. Ontology. In *http://en.wikipedia.org/wiki/Ontology*, 2012.