Planning with a Task Modeling Framework in Manufacturing Robotics

Jacob Huckaby[1] and Stavros Vassos[2] and Henrik I. Christensen[1]

Abstract— **In this paper we present the idea that by using AI planning in concert with formal task modeling, the overhead associated with plan creation for complex tasks can be reduced. The proposed approach uses a SysML taxonomy to model the system capabilities and the process specification, and the PDDL planning language to determine acceptable objective solutions. This idea is applied to the manufacturing domain, and examples are shown modeling a multi-robot system in an automobile manufacturing environment. A discussion is given regarding the merits of the demonstrated approach.**

I. Introduction

Robotics in manufacturing is seeing a second renaissance of sorts in today's world. This stems in a large part from recent advances in perception and manipulation platforms, and also a desire to free robots from their safety engineered cages to allow humans to work along side them to solve harder problems. Examples include efforts from Rethink Robotics (the Baxter platform) and ABB (the Frida platform), who are designing robots with flexible manipulation and advanced perception, that are specifically designed to work in collaboration with humans in human environments safely. With this renewed desire to automate tasks in the manufacturing domain, a problem that has persisted presents itself again: namely, the best way to model tasks, and to what extent can we automate the modeling of tasks and execution plans in these settings.

This problem applies to both these new areas where robots and humans are collaborating to solve tasks, but also to other areas of robotics and automation in manufacturing, such as large scale airplane wing assembly. One reason it has persisted for so long is simply that programming robots is hard. It requires deep technical knowledge, it is time consuming, and programming robots is done by specification with respect to the system requirements. For decades researchers in both industry and academia have looked for ways to reduce the overhead that this kind of automation entails. Methods such as programming by demonstration have attempted to reduce this load by teaching a robot tasks by demonstrating how those tasks are performed. Programming in this way requires less technical knowledge, is more intuitive, and would thus cut the necessary time required to program the robot.

Unfortunately, at least in manufacturing, these methods have not yet proven to be robust enough to generalize across a wide enough area such that they can be useful, and we are still left searching for a good way to automate task modeling and robot execution, such that these do not require users with specific technical skills and deep domain knowledge.

To be able to feasibly solve this problem, we first need a method to model the tasks necessary to enable the robot to achieve its programming objective. We begin here by specifying a taxonomy that defines skill primitives, or robot-specific skills that a robot can perform such as pick-up or detect, and constraints related to those skill primitives. Having this taxonomy allows us to specify how task components can be used to build higher-level tasks.

We proceed further to also show how artificial intelligence planning methods can be applied to the manufacturing assembly domain using this task modeling framework. We expect this work will contribute to the manufacturing robotics domain by showing how such a framework can be used in conjunction with formal planning methods to improve the automation process and reduce dependence on user effort. To the planning community, we expect this work to contribute to the body of knowledge by demonstrating relatively new planning methods, taking into account trajectory constraints, successfully applied to specific manufacturing problems.

This paper will proceed as follows. An overview of related work in robotics and manufacturing will be presented in section II. Following this, an introduction to our task modeling method, classical planning, and the BasicAssembly planning domain used in this work will be given in section III. Sections IV and V will present different examples of planning with manufacturing problems, along with a discussion of different design choices. We will conclude in section VI by mentioning future directions for this work.

II. Related Work

Given that programming robots is a difficult task requiring considerable technical knowledge, a great deal of prior work has focused on addressing how to make robot programming more intuitive and more accessible to operators with less technical experience. Much of this work has focused on finding efficient and effective methods for representing system knowledge to accomplish this task.

One philosophy has worked on encoding task knowledge as a function of motion. Examples of this type of representation include dynamical systems [10] and object-action complexes [14]. Another philosophy has espoused the view that one can effectively represent important system knowledge symbolically, such as topological task graphs [1]. This symbolic approach assumes that the system has more inherent knowledge (it knows how the relationships between the symbols and physical instantiations behave), while it allows for the modeling of more high-level concepts than

[1]Center for Robotics & Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA {huckaby,hic}@gatech.edu

[2]Department of Computer, Control, and Management Engineering, Sapienza University of Rome, 00185 Rome, Italy vassos@dis.uniroma1.it

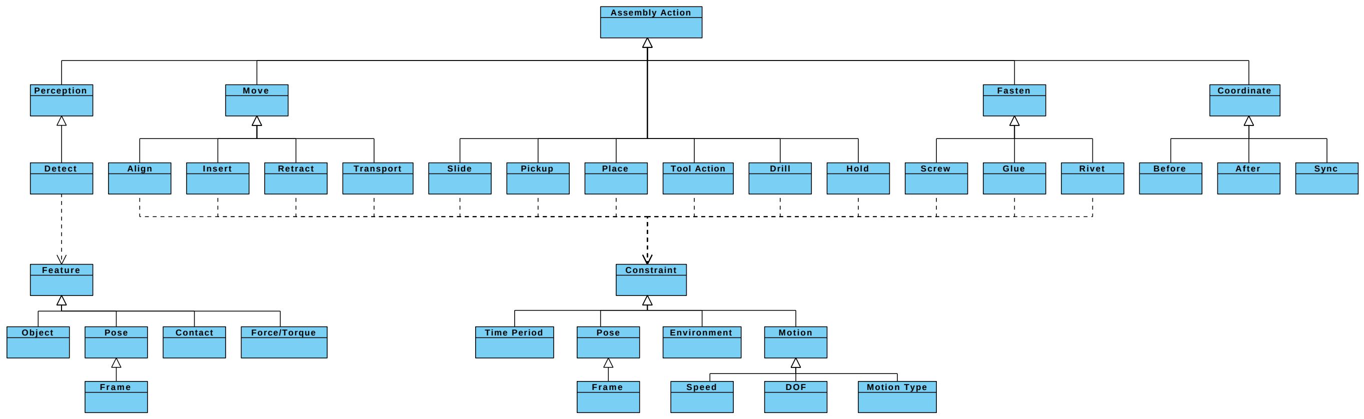

Fig. 1: A model instance for assembly tasks.

motion-based representations. Recent work by Kaelbling and Lozano-Pérez [11] attempts to combine symbolic task planning and geometric motion planning into a single approach.

More similar to our approach, other work has used different knowledge representations to simplify the robot programming problem. The work of Lyons et al. [15] defined a model for robot computation using port automata. Kosecka et al. [12] used a discrete event systems framework to model tasks and behaviors for robotics. Recent work includes that of Dantam, which takes a grammar-based approach to represent sensorimotor information [4].

There is also work that has been done in representing manufacturing and assembly objectives, such as the application of Petri Nets [19]. Another approach is the work of de Mello and Sanderson [8], which uses AND/OR graphs to enumerate all possible paths through the assembly process to get to the overall objective (e.g. an assembled product.) The paper then proposes to use a graph search algorithm to find an appropriate path through the graph based on specific problem specifications. However, this method does not allow for various goal specifications. Symbolic planning methods such as the one we adopt in this paper are able to provide a broad range goal specification based on sub-goals, trajectory constraints, cost metrics, and soft preferences.

Work by Kress-Gazit, et al., [13], [6] uses linear temporal logic to model task specifications to produce correct robot controllers for different tasks. While this planning representation is able to take advantage of the larger range of goal specifications as opposed to the previous example, it is far more complex than the planning representation used in our work, and perhaps is less likely to have a big impact to the manufacturing domain at this stage.

Similar to our goal, Balakirsky et al. [2] used the OWL ontology to provide a method for structuring knowledge in such a way as to be reusable for different problems, applied to kitting applications. Our work likewise proposes a multi-layered representation, but at a different level of abstraction. Our work also differs fundamentally in that our representation is proposed to improve not only modularity in knowledge, but also usability and intuitiveness for users.

III. System

A. Models, processes, and sequences for manufacturing

One goal of this work is to structure knowledge in such a way that lends itself to modularity, such that it can be used in different problems and contexts. To this end, we define the *model space* as the collection of all possible *capabilities* that a robot or robotic system may employ to accomplish some objective in the real world. An instance of model space would model the capabilities required to accomplish tasks in a specific application domain, such as manufacturing, laboratory automation, etc. The model space would thus span all capabilities needed in all robot application domains.

In this work, a *model instance* takes the form of a *SysML taxonomy* [17] that specifies each of these capabilities. SysML (or Systems Model Language) is a general modeling language for systems and systems engineering, and is defined as an extension of the popular UML modeling language. Benefits of using SysML include an expressive modeling language and tools for code generation, verification, and validation of system models.

For example, Figure 1 shows a model space instance for the manufacturing assembly domain [9]. The simplest building blocks in the taxonomy are the *skill primitives*, which are atomic actions that a robot is able to perform. The skill primitives are complemented by certain parameters or constraints on the action to be performed. For example, actions such as ***Align*** and ***Transport*** will often be done with respect to some goal or destination pose.

Based on the capabilities specified in the model space, we further define the *process space* as the collection of all system configurations and goal specifications that specify an environmental robot setup and objective. As with model space, a *process instance* takes the form of a SysML taxonomy, and it specifies the robot configurations and goals for a desired application domain. Figure 2 shows an instance for the manufacturing assembly domain.

Using these two instances, we can fully specify a *system* and *objective*. At this point, various methods can be used to find a solution or plan to solve this problem. One method of realizing this plan is the direct method, or having a human

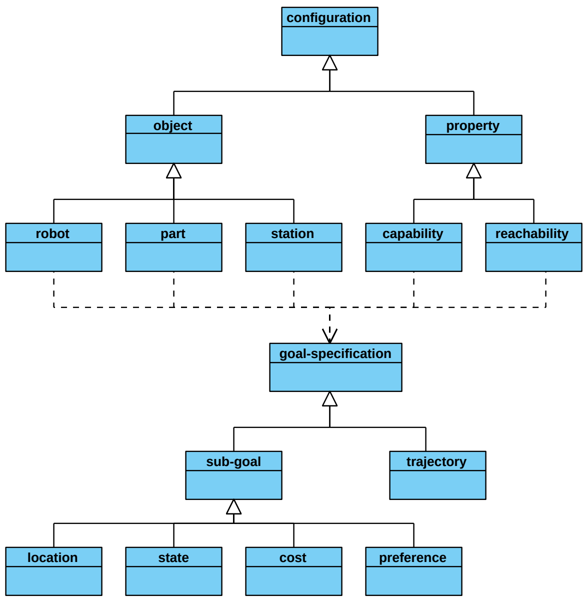

Fig. 2: A process instance for an assembly task.

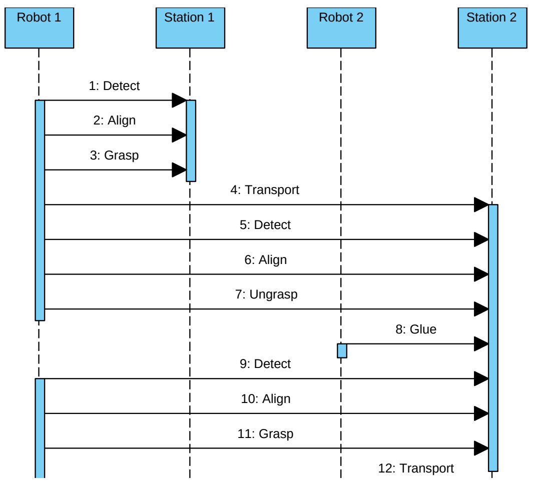

Fig. 3: A sequence instance for an assembly task.

directly encode the plan into SysML, which would then specify how the available robots would execute the actions to solve the problem. In this case the plan takes the form of a SysML *sequence instance*, which uses the hardware dependent implementation of skills to instruct the robot how to go about solving the problem, as for instance in Figure 3.

The sequence instance is intended to show the organization of an assembly objective into a *sequence of actions* that are to be performed, utilizing the skills described in the model taxonomy as discrete steps in the sequence. Each message from the robot to either the fixture or part bin represent an instantiation of a skill primitive (e.g., message 1 is a ***Detect***).

This manual encoding method can be effective, but has several drawbacks. As mentioned in the introduction, programming robots is hard as it requires deep technical knowledge and is time consuming. Apart from the overhead of specifying an initial procedure, one obvious difficulty is that any time some change in the configuration (or the target outcome) is required, additional resources are needed in order to specify an updated procedure.

Moreover, as this approach typically does not rely on a formal specification of the preconditions and effects of actions, no automated way can be used to check for errors and inconsistencies in the plan. This makes it possible that a user may even design a plan that could damage a workpiece, or put a robot into a dangerous configuration. Similarly, there is no easy way to guarantee success of the procedure in terms of meeting all of the objective goals or identify any guarantees on the optimality of the procedure. All of these issues are typically resolved by trial and error or using domain-specific tools for simulating and analyzing execution.

This is where artificial intelligence planning methods can offer significant benefits. By integrating planning into the workflow, the *planner* can take a considerable burden off of the user by addressing the previous issues. In this way parametrized solutions can be generated by the system, reducing the overhead required for manually specifying complex manufacturing tasks. Alternatively, the planner can aid the user by presenting potential plans that the user can then further tailor to meet changing needs in dynamic scenarios. In this work we aim for such functionality and adopt the so-called *Planning Domain Definition Language (PDDL)* [16].

The idea is that based on a *model instance* that corresponds to a particular manufacturing domain, we can specify a *PDDL planning domain* that formalizes the preconditions and effects of the available actions in the domain. Then, from a *process instance* that corresponds to a particular manufacturing problem in this domain, we can specify a *PDDL planning problem* that formalizes the configuration of the available robots and resources as well as the intended goal we want to achieve. Then with these two specifications at hand we can employ *PDDL planners* to search and find a solution, that is, in effect an appropriate *sequence instance* for this manufacturing domain/problem combination.

As the academic field of automated planning evolves, this approach to generating sequence instances in an automated way can also enjoy the increasing set of features for the developed PDDL planners. In particular, apart from generating a sequence instance that satisfies basic requirements, state-of-the-art planners can take into account action cost, duration, parallelism, and are able to search for optimized solutions that maximize or minimize given metrics and preferences.

Next, we introduce the necessary details of STRIPS planning and PDDL, and proceed to specify a PDDL planning domain for the assembly manufacturing domain.

B. STRIPS planning and the language of PDDL

In the area of classical planning one is faced with the following task: given i) a complete specification of the initial state of the world, i.e., an application domain, ii) a set of action schemas that describe how the world may change, and iii) a goal condition, one has to find a sequence of actions such that when applied one after the other in the initial state, they transform the state into one that satisfies the goal condition. In this paper we focus on the STRIPS formalism [5] with some extensions.

In STRIPS, the representation of the initial state, the action schemas, and the goal condition is based on literals from predicate logic. For example, $\textit{arm-at}(\textit{arm}_3, \textit{station}_7)$ is a positive literal that may be used to represent that the robotic arm "3" is located at the working station "7". Similarly, the negative literal $\neg\textit{arm-holding}(\textit{arm}_3, \textit{part}_{12})$ may be used to say that the same arm is not holding a particular part.

The *initial state* is specified as a set of positive ground literals. This provides a complete specification of the state based on a closed-world assumption, that is, for all ground literals not included in the set, the negative version of the literal is assumed to hold.

The *action schemas* specify the available actions as well as their preconditions and effects using sets of ground literals. In the case of preconditions a set of positive literals specify what needs to be present in the state representation in order for the action to be executable, and for the effects of an action, a set of positive and negative literals specify how the state should be transformed after action execution: all the positive literals in the set of effects are added in the set describing the state, and all negative literals are removed.

A *goal condition* is also a set of positive ground literals. The intuition is that the goal is satisfied if all the literals listed in the goal condition are included in the set that describes the state. A solution then to a planning problem is a sequence of actions such that if they are executed starting from the initial state, checking for corresponding preconditions, and applying the effects of each action one after the other, they lead to a state satisfying the goal condition.

In this paper we focus on a specific syntax for describing STRIPS planning tasks following the Planning Domain Definition Language (PDDL) [16]. PDDL is a standard language for specifying planning tasks that is widely used in the academic planning community. In PDDL, the specification of the predicates and the action schemas is separated from the specification of the initial state and the goal condition. The first part is typically referred to as the *planning domain* and the second part as the *planning problem*. This allows us to define a number of planning problems for the same planning domain. In particular, for a manufacturing application domain a planning domain can specify the functionalities of robots that may be available at different times. Using this planning domain then, planning problems can represent different configurations for the available robots and a goal condition for the corresponding manufacturing task.

The syntax of PDDL follows the logical representation of literals but a prefix notation is used. In this way, the positive literal $\textit{arm-holding}(a, p)$ may be represented as `(arm-holding ?a ?p)`, and the negative version of the literal may be represented as `(not(arm-holding ?a ?p))`. Note that variables are denoted using a preceding question mark. The special predicates `(:predicates ...)` and `(:action ...)` are used for the specification of the planning domain, with the intuitive meaning. Similarly, the special predicates `(:objects ...)`, `(:init ...)`, and `(:goal ...)` are used to specify a planning problem. This notation will become more clear in the next section where we introduce the BasicAssembly planning domain.

Finally, we will appeal to some more advanced features of PDDL that go beyond STRIPS. In particular we will adopt *types* for objects following functionality ADL [18]. For example `?a - arm` will be used to denote that variable `?a` can only be replaced by an object of type `arm`. Moreover, we will appeal to PDDL3 [7], one of the latest specifications of the PDDL language, in order to specify simple *trajectory constraints* for the desired goal condition. These constraints will be used to specify the intended order in which the sub-goals of the manufacturing task need to be executed, essentially specifying the steps of a process to be planned.

C. The BasicAssembly planning domain

In this section we present the BasicAssembly PDDL planning domain that captures some of the basic capabilities of manufacturing robots. In particular, the domain represents the high-level actions that are abstracted in the Capabilities SysML taxonomy described in the previous section. These actions are represented in terms of their preconditions and effects. The intention is that in this way a system can plan on the level of these capabilities of robots, leaving lower-level activities such as motion planning to be handled at a lower level at run-time.

A manufacturing site is represented as a set of robotic *arms* that abstract the sensors and actuators of the site, a set of *parts* that abstract the pieces to be handled in order to achieve the relevant manufacturing task, and a set of *stations*, each of which may be occupied by one or more robotic arms and may be used to place or hold a number of parts. Except for the very basic actions of moving and grasping that are modeled with a separate action, a set of *tools* represents the available operations that can be performed on the parts. In PDDL terms, arms, parts, stations, and tools are all objects of the PDDL problem. Following the typed-approach though, there are four basic types in order to distinguish between them, namely types `arm`, `part`, `station`, and `tool`.

We assume that the low-level details required for an arm interacting with parts and stations is abstracted by two high-level "detect-pose" actions, which can be handled appropriately by each robotic arm at execution time. In the high-level representation of BasicAssembly, in order for a robotic arm to pick up a part `?p` located at station `?s`, it should first move to station `?s`, perform a detect-pose action using available sensors, and then grasp the part using an attached gripper (provided the arm is equipped with one).

The BasicAssembly PDDL domain uses a number of predicates and action schemas to represent some basic functionalities in manufacturing domains, as explained next.

a) The predicates of the domain:

The predicates of the BasicAssembly planning domain essentially represent properties of the state of the manufacturing site at any time. In particular, the following predicates represent information regarding the available robotic arms.

- (arm-canreach ?a - arm ?s - station)
- (arm-at ?a - arm ?s - station)
- (arm-capabilities ?a - arm ?t - tool)
- (arm-active ?a - arm ?t - tool)
- (arm-holding ?a - arm ?p - part)

The following predicates represent properties of the parts. Note that some literals are used as book-keeping information for modeling the robotic arm actions and their effects.

- (part-at ?p - part ?s - station)
- (part-state ?p - part ?t - tool)
- (pose-detected ?a - arm ?p - part ?s - station)

More details for these predicates and their use can be found in the complete PDDL specification of the BasicAssembly domain in the appendix.

The intuition is that a set of positive ground literals of this kind can be used to represent the initial state of the manufacturing site and the goal condition for planning purposes. For example, the positive ground literals (arm-canreach arm1 st1) and (arm-canreach arm1 st2) can be used to model that arm1 can reach both stations st1 and st2 (and in fact according to the closed world assumption *only those stations*). Similarly, (arm-capabilities arm1 grip) may be used to model that arm1 has a gripper.

We now proceed to present the action schemas that characterize the available actions for the robotic arms.

b) The actions of the domain:

Action schemas in PDDL are defined in terms of their preconditions and effects using the predicates of the domain. For example, the following statement is an action schema of the BasicAssembly domain that specifies the action of a robotic arm ?a grasping a part ?p from station ?s.

```
(:action grasp
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and
    (arm-at ?a ?s)
    (part-at ?p ?s)
    (arm-active ?a grip)
    (arm-holding ?a no-part)
    (pose-detected ?a ?p ?s))
  :effect (and
    (arm-holding ?a ?p)
    (not (arm-holding ?a no-part))
    (not (part-at ?p ?s))
    (not (pose-detected ?a ?p ?s)))
)
```

The intuition is that a ground instance of this action schema, e.g., (grasp arm1 p station2), can be performed when the arm is located at the same station as the part, the grip tool is activated, the arm is not holding another part, and the required information about pose detection is already present. These requirements can be met by the previous execution of other appropriate actions, in particular one that moves the arm to the right station, one that activates the gripper, and one that provides information about the low-level pose to be realized. These actions are also part of the BasicAssembly domain and will be described next.

The result of performing the action is that the relevant part is no longer located at the station, therefore the corresponding literal is removed from the description of the state, while a positive literal is added to represent that the arm is now holding the part, among a couple of other details.

Note that in order to allow maximum compatibility with available planners we use only positive literals in the preconditions of actions, and as a result we model the information that a robotic arm ?a is not holding any part with the literal (arm-holding ?a no-part). This could be done also with another predicate of the form (arm-gripperfree ?a) but we chose to use the same predicate arm-holding for uniformity. Note also that we abstract the pose detection and alignment needed, using an appropriate high-level condition concerning the arm, part, and station in question, that is, (pose-detected ?a ?p ?s) that can be achieved by an appropriate detect action as explained next.

We now give a list of the action schemas of the BasicAssembly domain which have the intuitive meaning. The details for each action schema (parameters, preconditions, effects) can be found in the complete PDDL specification of the BasicAssembly domain in the appendix.

- activate(?a -arm ?old - tool ?new - tool)
- grasp(?a - arm ?p - part ?s - station)
- ungrasp(?a - arm ?p - part ?s - station)
- move(?a - arm ?from - station ?to - station)
- carry(?a - arm ?p - part ?from - station ?to - station)
- employ(?a - arm ?t - tool ?p - part ?s - station)
- detect-pose-part(?a - arm ?p - part ?s - station)
- detect-pose-station(?a - arm ?p - part ?s - station)

Note that in order to allow maximum compatibility with available planners we also avoided using conditional effects, and as a result chose to model the movement of an arm by two different actions, one when the gripper is empty (move) and one when the arm holds some part (carry).

IV. A CONCRETE EXAMPLE USING BASICASSEMBLY

In this section we will present an application example that demonstrates the use of the BasicAssembly planning domain.

A. Car-door manufacturing scenario

This task concerns the preparation of a car-door part for assembly into a car, taken directly from the factory floor as shown in Figure 4. We will present two variations on the common scenario according to which the part must be glued and then welded before it is added to the assembly line.

B. Process instance P1

The first configuration of the scenario is shown in Figure 5. There are four stations: the part bin (the originating location for the parts to be worked on), work station 1 (the gluing station), work station 2 (the welding station) and the assembly line. There are three robots, one which is able to

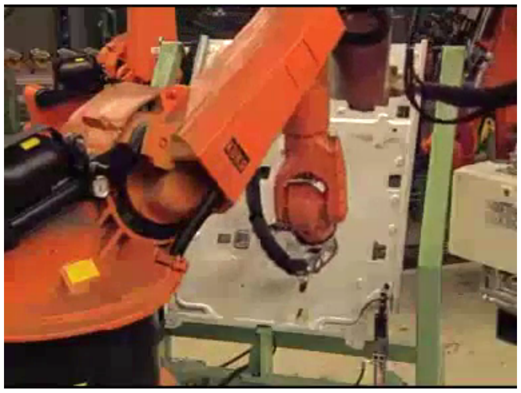
Fig. 4: Manufacturing of car door.

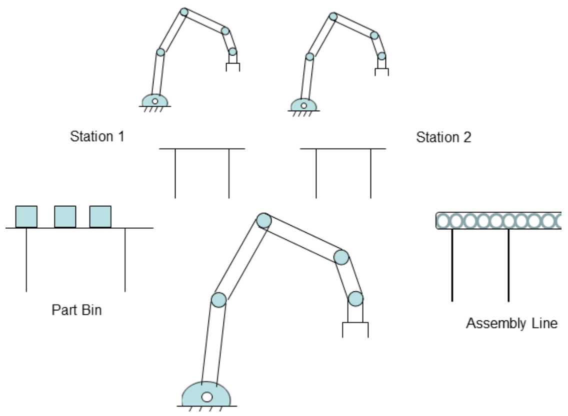

Fig. 5: Assembly process instance P1.

reach all stations, and two specialized robots for each work station. This problem is formalized in PDDL as follows:

```
(define (problem p1) (:domain assembly)
  (:objects
    arm1 arm2 arm3 - arm
    car-door no-part - part
    part-bin station1 station2 assembly-line - station
    grip detect weld glue no-tool - tool )

  (:init
    (arm-canreach arm1 part-bin)
    (arm-canreach arm1 station1)
    (arm-canreach arm1 station2)
    (arm-canreach arm1 assembly-line)
    (arm-canreach arm2 station1)
    (arm-canreach arm3 station2)
    (arm-at arm1 part-bin)
    (arm-at arm2 station1)
    (arm-at arm3 station2)
    (arm-capabilities arm1 grip)
    (arm-capabilities arm1 detect)
    (arm-capabilities arm2 glue)
    (arm-capabilities arm2 detect)
    (arm-capabilities arm3 weld)
    (arm-capabilities arm3 detect)
    (arm-active arm1 no-tool)
    (arm-active arm2 no-tool)
    (arm-active arm3 no-tool)
    (arm-holding arm1 no-part)
    (arm-holding arm2 no-part)
    (arm-holding arm3 no-part)
    (part-at car-door part-bin))

  (:goal
    (and (part-state car-door glue)
         (part-state car-door weld)
         (part-at car-door assembly-line)))

  (:constraints
    (and (sometime-before (part-state car-door weld)
         (part-state car-door glue))
              (sometime-before (part-at car-door assembly-line)
         (part-state car-door weld)))))
```

Predicate `:objects` is used to specify all the available objects in the PDDL problem (corresponding to the process instance in question). Note that objects are listed according to their type, e.g., here there objects of type `arm` are specified.

Predicates `:init` and `:goal` specify the initial state and the desired final state, using positive ground literals. The initial state is specified as a list of literals, while the goal condition is formalized as a logical sentence (in this case using logical conjunction). The intuition is that the final state should be such that all three sub-goals hold, i.e., the glue tool has been employed to the car-door, the weld tool also, and the car-door is located at the assembly line.

Note that in basic STRIPS planning problems there is no information about the order in which the sub-goals need to be achieved. Extensions, though, have investigated how to fine-tune the resulting plan in terms of soft preferences and hard constraints. In particular in PDDL3, one of the latest versions of PDDL, hard trajectory constraints can be used to specify the order in which sub-goals should be achieved. These are specified using the `:constraints` predicate and the `sometime-before` keyword.

Trajectory constraints are a crucial aspect in the context of manufacturing as essentially we are interested in planning for a particular type of sequence, not just any one that achieves sub-goals in any order. As this feature is a recent addition, currently not many planners fully support it. For our experiments we used the planner OPTIC [3] which was one of the very few able to handle trajectory constraints well and also supports other features we intend to use.

The resulting plan is shown here in time-steps:

```
01: (activate arm1 grip detect)
01: (activate arm2 glue detect)
01: (activate arm3 weld detect)
02: (detect-pose-part arm1 car-door part-bin)
03: (activate arm1 detect grip)
04: (grasp arm1 car-door part-bin)
05: (carry arm1 car-door part-bin station2)
05: (activate arm1 grip detect)
06: (detect-pose-station arm1 car-door station2)
07: (carry arm1 car-door station2 station1)
08: (detect-pose-station arm1 car-door station1)
09: (carry arm1 car-door station1 assembly-line)
10: (detect-pose-station arm1 car-door assembly-line)
11: (activate arm1 detect grip)
11: (carry arm1 car-door assembly-line station1)
12: (ungrasp arm1 car-door station1)
13: (detect-pose-part arm2 car-door station1)
13: (activate arm1 grip detect)
14: (activate arm2 detect glue)
14: (detect-pose-part arm1 car-door station1)
15: (employ arm2 glue car-door station1)
15: (activate arm1 detect grip)
16: (grasp arm1 car-door station1)
17: (carry arm1 car-door station1 station2)
18: (ungrasp arm1 car-door station2)
19: (detect-pose-part arm3 car-door station2)
19: (activate arm1 grip detect)
20: (activate arm3 detect weld)
20: (detect-pose-part arm1 car-door station2)
21: (employ arm3 weld car-door station2)
21: (activate arm1 detect grip)
22: (grasp arm1 car-door station2)
23: (carry arm1 car-door station2 assembly-line)
24: (ungrasp arm1 car-door assembly-line)
```

Note that the planner accounts for parallelism as some actions of different arms occur at the same time, e.g., the activation of tools for all three arms at time-step 01.

C. Process instance P2

Now we report on a variation that is similar to the previous process instance, except that this time robot arm1 is not able to reach station2. Instead, the specialized robot arm2 is able to reach both station1 and station2. This implies that moving the car-door part from station1 to station2 to station3 cannot be done by arm1 alone as in the previous case. Instead, an additional coordination between arm1 and arm2 is required. All other considerations are the same as the previous example. The resulting plan follows:

```
01: (activate arm1 grip detect)
01: (activate arm2 grip detect)
01: (activate arm3 weld detect)
02: (detect-pose-part arm1 car-door part-bin)
03: (activate arm1 detect grip)
04: (grasp arm1 car-door part-bin)
05: (carry arm1 car-door part-bin station1)
05: (activate arm1 grip detect)
06: (detect-pose-station arm1 car-door station1)
07: (carry arm1 car-door station1 assembly-line)
08: (detect-pose-station arm1 car-door assembly-line)
09: (activate arm1 detect grip)
09: (carry arm1 car-door assembly-line station1)
10: (ungrasp arm1 car-door station1)
11: (activate arm1 grip detect)
11: (detect-pose-part arm2 car-door station1)
12: (detect-pose-part arm1 car-door station1)
12: (activate arm2 detect glue)
13: (activate arm1 detect grip)
13: (employ arm2 glue car-door station1)
14: (activate arm2 glue grip)
15: (activate arm2 grip detect)
16: (detect-pose-part arm2 car-door station1)
17: (activate arm2 detect grip)
18: (grasp arm2 car-door station1)
19: (activate arm2 grip detect)
20: (detect-pose-station arm2 car-door station1)
21: (carry arm2 car-door station1 station2)
22: (detect-pose-station arm2 car-door station2)
23: (activate arm2 detect grip)
24: (ungrasp arm2 car-door station2)
25: (detect-pose-part arm3 car-door station2)
25: (activate arm2 grip detect)
26: (activate arm3 detect weld)
26: (detect-pose-part arm2 car-door station2)
27: (employ arm3 weld car-door station2)
27: (activate arm2 detect grip)
28: (grasp arm2 car-door station2)
29: (carry arm2 car-door station2 station1)
30: (ungrasp arm2 car-door station1)
31: (grasp arm1 car-door station1)
32: (carry arm1 car-door station1 assembly-line)
33: (ungrasp arm1 car-door assembly-line)
```

Note that this process instance needs more time-steps than the previous one. This variation demonstrates the planner's ability to deal with configuration and process changes.

V. Discussion

In this work we illustrate the utility of using a formal planning domain specification coupled with a task modeling framework. The simple example presented highlights some of the basic aspects of what can be achieved.

As a first step we focused on a high-level representation of the preconditions and effects of actions in a manufacturing assembly domain, that can be used to provide well-structured solutions in an automated way. These solutions still require manual work in a lower level in order to connect the high-level abstract actions to the low-level capabilities of each component. In the general case of manual robot programming, this is difficult and requires resources such as time and technical knowledge. When the scenario is updated or changed, the manual robot programmer must reprogram the entire scenario from the ground up, even for small changes. The intention is that by relying on an appropriate abstraction for robotic actions and realizing the high-level/low-level connection for the available hardware, one can benefit by re-using these models to handle future scenarios, and reduce the resources required to realize the solutions.

It should be noted that a number of assumptions go into this approach as currently implemented. The lowest level of abstraction relies on hardware-specific implementations such as motion planners and pose estimation algorithms. The current system does not take fault detection or recovery into consideration, and assumes that actions proposed by the system are successfully achieved by the robot. Other assumptions have to do with the environment, or process model. For example, we believe it is not unreasonable to assume that every workpiece in the environment is reachable, as in manufacturing settings the environment is often highly engineered to make the task achievable. Note also that this approach will not significantly speed up some aspects of programming the system, such as specifying complex kinematic motions (the weld trajectory, for example). This is to be expected, however, as this approach has targeted modeling and generation of manufacturing assembly solutions, not methods of parameterization.

One direction for future work is to investigate more refined abstractions that can provide a variable level-of-detail for available hardware platforms used in practice. For example, it may be helpful (or for some platforms necessary) to distinguish as two separate actions the task of detecting an object and detecting a location pose for placing that object. Also, different types of robotic devices may be able to handle an action set that spans more than one level-of-detail. The intuition is that as the action representations become more specific modeling the capabilities of existing platforms, the planner obtains more detailed information and can then provide solutions of better quality that can be incorporated easier in the real manufacturing setting.

A different direction for future work is incorporating more features for refining the desired solution for a sequence instance. PDDL3 already accounts for cost metrics, preferences, as well as continuous time and durative actions. We intend to investigate which combination of features is intuitive and practical and can be effective in providing greater flexibility by automatically generating solutions as sequences for complex manufacturing problems.

Finally, our experimentation with available PDDL planners showed that even though many advanced features have been investigated leading to robust solutions, there is less focus on trajectory constraints, which is the most crucial one for the type of problems we are interested in the manufacturing setting. We believe that our work may provide useful feedback to the planning community with respect to a possible wide application domain.

VI. Conclusion

In this paper we investigate how a planning domain representation in concert with a task modeling framework can be used to reduce the work involved in plan creation for complex manufacturing tasks. This provides the opportunity to relieve some burden from the robot user, and to improve the reliability of those plans. For the scope of this paper we focused on basic functionality of robotic arms in a manufacturing assembly setting. Our investigation shows that recent versions of the so-called Planning Domain Definition Language (PDDL), and software planners conforming to them, are suitable for representing the relevant problems and searching for solutions in an automated way. For future work we intend to explore more complex manufacturing objectives, especially cases that require more varied specification using metrics and preferences.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of Boeing for funding this research.

REFERENCES

[1] T. Abbas and B.A. MacDonald. Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstration (pbd) processes. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3816 –3821, may 2011.

[2] Stephen Balakirsky, Zeid Kootbally, Craig Schlenoff, Thomas Kramer, and Satyandra Gupta. An industrial robotic knowledge representation for kit building applications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, 2012.

[3] J. Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs, 2012.

[4] N. Dantam and M. Stilman. The motion grammar: Linguistic perception, planning, and control. In *Robotics: Science and Systems*, 2011.

[5] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[6] C. Finucane, Gangyuan Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1988 –1993, oct. 2010.

[7] A. Gerevini and D. Long. Preferences and soft constraints in PDDL3. *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, pages 46–53, 2006.

[8] L.S. Homem de Mello and A.C. Sanderson. And/or graph representation of assembly plans. *Robotics and Automation, IEEE Transactions on*, 6(2):188–199, Apr.

[9] Jacob Huckaby and Henrik I. Christensen. A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics. In *8th International Cognitive Robotics Workshop*, July 2012.

[10] A.J. Ijspeert, J. Nakanishi, T. Shibata, and S. Schaal. Nonlinear dynamical systems for imitation with humanoid robots. In *Proceedings of the IEEE International Conference on Humanoid Robots*, pages 219–226. 2001.

[11] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*, 2011. Finalist, Best Manipulation Paper Award.

[12] Jana Kosecka and Ruzena Bajcsy. Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12:187–198, 1993.

[13] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *Robotics Automation Magazine, IEEE*, 18(3):65 –74, sept. 2011.

[14] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann. Object-Action Complexes: Grounded Abstractions of Sensory-motor Processes. *Robotics & Autonomous Systems*, 59(10):740–757, 10 2011.

[15] D.M. Lyons and M.A. Arbib. A formal model of computation for sensory-based robotics. *Robotics and Automation, IEEE Transactions on*, 5(3):280 –293, jun 1989.

[16] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003, Yale Center for Computational Vision and Control, 1998.

[17] OMG. OMG Systems Modeling Language (OMG SysML) Version 1.2 Specification, 2010.

[18] Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 324–332, 1989.

[19] J Rosell. Assembly and task planning using petri nets: a survey. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 218(8):987–994, 2004.

APPENDIX

A. BasicAssembly PDDL domain

```
(define (domain assembly)
 (:requirements :strips :typing)

 (:types arm station tool part)

 (:constants
  no-tool grip detect - tool
  no-part - part )

 (:predicates
  (arm-canreach ?a - arm ?s - station)
  (arm-at ?a - arm ?s - station)
  (arm-capabilities ?a - arm ?t - tool)
  (arm-active ?a - arm ?t - tool)
  (arm-holding ?a - arm ?p - part)
  (part-at ?p - part ?s - station)
  (part-state ?p - part ?t - tool)
  (pose-detected ?a - arm ?p - part ?s - station) )

 (:action activate
  :parameters (?a -arm ?old - tool ?new - tool)
  :precondition (and (arm-capabilities ?a ?new)
                     (arm-active ?a ?old))
  :effect (and (arm-active ?a ?new) (not (arm-active ?a ?old))) )

 (:action detect-pose-part
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and (arm-at ?a ?s) (arm-active ?a detect)
                     (part-at ?p ?s))
  :effect (pose-detected ?a ?p ?s) )

 (:action detect-pose-station
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and (arm-at ?a ?s) (arm-active ?a detect)
                     (arm-holding ?a ?p))
  :effect (pose-detected ?a ?p ?s) )

 (:action grasp
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and (arm-at ?a ?s) (arm-active ?a grip)
        (arm-holding ?a no-part) (part-at ?p ?s)
        (pose-detected ?a ?p ?s))
  :effect (and (arm-holding ?a ?p) (not (arm-holding ?a no-part))
        (not (part-at ?p ?s)) (not (pose-detected ?a ?p ?s))) )

 (:action ungrasp
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and (arm-at ?a ?s) (arm-active ?a grip)
                     (arm-holding ?a ?p) (pose-detected ?a ?p ?s))
  :effect (and (arm-holding ?a no-part) (not (arm-holding ?a ?p))
        (part-at ?p ?s) (not (pose-detected ?a ?p ?s))) )

 (:action move
  :parameters (?a - arm ?from - station ?to - station)
  :precondition (and (arm-at ?a ?from) (arm-canreach ?a ?to)
        (arm-holding ?a no-part))
  :effect (and (arm-at ?a ?to) (not (arm-at ?a ?from))) )

 (:action carry
  :parameters (?a - arm ?p - part ?from - station ?to - station)
  :precondition (and (arm-at ?a ?from) (arm-canreach ?a ?to)
                     (arm-holding ?a ?p))
  :effect (and (arm-at ?a ?to) (not (arm-at ?a ?from))) )

 (:action employ
  :parameters (?a - arm ?t - tool ?p - part ?s - station)
  :precondition (and (arm-at ?a ?s) (arm-active ?a ?t)
                (arm-holding ?a no-part) (part-at ?p ?s)
                (pose-detected ?a ?p ?s))
  :effect (and (part-state ?p ?t) (not (pose-detected ?a ?p ?s))))
)
```