Robotics and Computer-Integrated Manufacturing 33 (2015) 21–28

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Ontology based action planning and verification for agile manufacturing

Stephen Balakirsky

Georgia Tech Research Institute, Atlanta, GA 30332, USA

ARTICLE INFO

Article history:
Received 30 April 2014
Received in revised form
24 July 2014
Accepted 21 August 2014
Available online 18 October 2014

Keywords:
Knowledge driven system
Adaptive planning
Manufacturing
Ontology
Robotics
Planning Domain Definition Language

ABSTRACT

Many of today's robotic work cells are unable to adapt to even small changes in tasking without significant reprogramming. This results in downtime for production lines anytime a change to a product or procedure must be made. This paper examines a novel knowledge-driven system that provides added agility by removing the programming burden for new activities from the robot and placing it in the knowledge representation. The system is able to automatically recognize and adapt to changes in its work-flow and dynamically change assignment details. The system also provides for action verification and late binding of action parameters, thus providing flexibility by allowing plans to adapt to production errors and changing environmental conditions. The key feature of this system is its knowledge base that contains the necessary relationships and representations to allow for adaptation. This paper presents the ontology that stores this knowledge as well as the overall system architecture. The manufacturing domain of kit construction is examined as a sample test environment.

1. Introduction

Many of today's robotic arms are capable of obtaining sub-millimeter accuracy and repeatability. Robots such as the Fanuc LR Mate 200*i*D claim ± 0.02 mm repeatability [1] which has been verified in various publicly viewable experiments [2,3]. However, these same systems lack the sensors and processing necessary to provide a representation of the work cell in which they reside or of the parts that they are working with. In fact, according to the International Federation of Robotics (IFR), over 95% of all robots in use do not have a sensor in the outer feedback loop. They rely on fixtures to allow them to be robust in the presence of uncertainty [4]. This lack of sensing in the outer feedback loop leads to systems that are taught or programmed to provide specific patterns of motion in structured work cells over long production runs. These systems are unable to detect that environmental changes have occurred, and are therefore unable to modify their behavior to provide continued correct operation.

Just-in-time manufacturing and small batch processing require changes in the manufacturing process on a batch-by-batch or an item-by-item basis. This leads to a reduction in the number of cycles that a particular pattern of motion is useful and increases the percentage of time necessary for robot teaching or programming over actual cell operation. This teaching/programming time requires that the cell be taken off-line which greatly impacts productivity. For small batch processors or other customers who must frequently change their line configuration, this frequent downtime and lack of adaptability may be unacceptable.

Research aimed at increasing a robot's knowledge and intelligence has been performed to address some of these issues. It is anticipated that proper application of this intelligence will lead to more agile and flexible robotic systems. Both Huckaby et al. [5] and Pfrommer et al. [6] have examined the enumeration of basic robotic skills that may be dynamically combined to achieve production goals. The EU-funded RObot control for Skilled ExecuTion of Tasks in natural interaction with humans (ROSETTA) [7] and Skill-Based Inspection and Assembly for Reconfigurable Automation Systems (SIARAS) [8] have proposed distributed knowledge stores that contain representations of robotic knowledge and skills. The focus of these programs is to simplify interaction between the user and the robotized automation system.

The IEEE Robotics and Automation Society's Ontologies for Robotics and Automation Working Group [9] has also taken the first steps in creating a knowledge repository that will allow greater intelligence to be resident on robotic platforms. The Industrial Subgroup of this working group has applied this infrastructure to create a sample kit building system. Kit building may be viewed as a simple, but relevant manufacturing process.

Balakirsky et al. [10] describe a kitting system based on the IEEE knowledge framework that allows greater flexibility and agility by utilizing a Planning Domain Definition Language (PDDL) [11] planning system to dynamically alter the system's operation in order to adapt to variations in its anticipated work flow. The

E-mail address: stephen.balakirsky@gtri.gatech.edu

http://dx.doi.org/10.1016/j.rcim.2014.08.011
0736-5845/© 2014 Elsevier Ltd. All rights reserved.

system does not require *a priori* information on part locations (i.e. fixturing is not required) and is able to build new kit varieties without altering the robot's programming. While this body of work makes great strides in removing the need to teach/program the robot between production runs, all of the PDDL predicates and actions must still be programmed/taught.

This means that any modification to the production process that requires a new PDDL predicate or action will still require that the production line be brought down for programming/teaching. The next logical step in adding agility to robotic production is to remove this programming/teaching step when new actions and predicates are required by the system. This body of work examines utilizing a basis set of robotic primitive actions to enumerate new robotic skills and the enhancement of the IEEE ontology to store those skills in a reusable knowledge store. This removes the need to program the robot for new predicates and actions. The sample domain of kit building is utilized to demonstrate this work.

The organization of the remainder of this paper is as follows. Section 2 provides an overview of the PDDL language and a discussion of how PDDL is integrated into the ontology. Section 3 discusses the detailed operation of the cell and presents the system's architecture, and Section 4 discusses the knowledge representation and the ontology. Finally, Section 5 presents conclusions and future work.

2. PDDL

The objective behind domain independent planning is to formulate a planner that is able to construct plans across a wide variety of planning domains with no change to the planning algorithms. The typical problem presented to such a planner consists of

- a set of objects,
- a set of predicate expressions that define properties of objects and relations between objects,
- a set of actions that are able to manipulate the predicate expressions,
- a set of predicate expressions that constitute a partial world state and make up the initial conditions,
- a set of problem goals, which are predicate expressions that are required to be true at the end of a plan.

If an *action* is defined as a fully instantiated operator, then the job of the planner is to formulate a sequential list of valid actions, referred to as a *valid plan*, which will bring the system from the state represented by the initial conditions to a state that satisfies the problem goals (all of the problem goals are simultaneously true).

PDDL is designed as a standard language and structure for representing a valid plan along with all of the elements of domain independent planning systems. Fig. 1 depicts a schema view of our augmented PDDL representation. As in a standard PDDL representation, a set of object *types* is represented along with *predicate* expressions and *actions*. The schema has been augmented with the representation of *functionalPredicates* and the distinction between *RobotActions* and *VisionActions*.

In PDDL, *types* must be declared before their use as parameters to predicate expressions. For our PDDL extension, the additional requirement has been added that all types must be defined in the system's ontology and must be derived from the base *SolidObject* or *SystemConstant* classes. This assures that basic properties of objects being used as parameters are known to the system. The *SolidObject* provides the basis for all physical objects in the world while the *SystemConstant* represents a named system memory

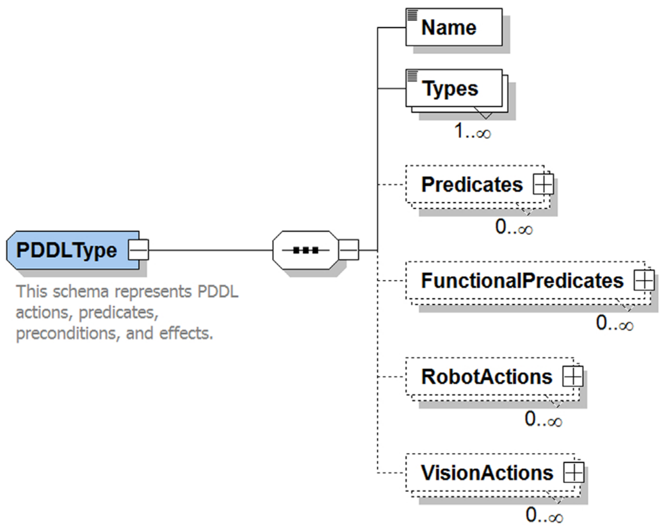

Fig. 1. Description of the PDDLType class that is designed as an augmented PDDL description language. It contains all of the information necessary for interacting with the robot cell's robot controller and vision system.

location that may be used as intermediate storage of values between commands. It is often used by *VisionActions* to store values required by a future *RobotAction*.

Predicates are binary expressions that contain one or two objects of defined *types* as arguments and provide a partial definition of the world's state. Predicates may be used for *preconditions* (predicates that must be true for an action to be executed) as well as *effects* (predicates that are expected to become true as the result of an action). An additional class of predicates known as *FunctionalPredicates* has been added to this representation. These predicates allow for mathematical operations to be performed between parameters. For example, the predicate *equalTo*(*obj1*, *value*) will evaluate the equivalence between *obj1* and *value* while the predicate *set-value*(*systemVariable*, *setValue*, *valueType*) will set the value of the variable *systemVariable* to *setValue*. The *set-value* predicate will evaluate to be *true* if the memory location to be set exists and the value to be set is of the correct type for that memory location.

Actions represent compound tasks that the robot cell must accomplish. Our robot cell consists of a robotic arm and a vision system. Therefore, our actions have been segregated into *RobotActions* that pertain to the robot system and *VisionActions* that pertain to the vision system. More information on the implementation of these types in the ontology may be found in Section 4.

3. System operation

The framework that has been implemented as part of this work is a deliberative intelligent system based on a single level or echelon of the hierarchal 4D/RCS reference model architecture [12] and is tightly coupled with a domain independent planning system. As shown in Fig. 2, 4D/RCS follows a sense-model-act paradigm.

A central feature of 4D/RCS is its world model. As shown in Fig. 3, the world model for this system may be decomposed into the three parts: Reasoning, Planning, and Execution. All of the concepts necessary for the industrial domain under test and for PDDL plan execution are encoded in the ontology that resides in the reasoning section of the model. The planning and execution sections of the model are automatically generated from this section.