Elsevier Editorial System(tm) for Robotics and Computer Integrated Manufacturing
Manuscript Draft

Manuscript Number: RCIM-D-14-00111R1

Title: Ontology Based Action Planning and Verication for Agile Manufacturing

Article Type: SI:Knowledge Driven Robotics

Keywords: knowledge driven system; adaptive planning; manufacturing; ontology; robotics; Planning Domain Definition Language

Corresponding Author: Dr. Stephen Balakirsky,

Corresponding Author's Institution: Georgia Tech Research Institute

First Author: Stephen Balakirsky

Order of Authors: Stephen Balakirsky

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 article 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 article 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.

***Cover Letter**

April 30, 2014

Georgia Institute of Technology

Please find my submission to the special issue on Knowledge Driven Robotics enclosed. Feel free to contact me with any questions that you may have. Note that the main .tex file is titled “Balakirsky RCIM 2014.tex”

Best,

Dr. Stephen Balakirsky
Senior Research Scientist
(404) 407-8547
Stephen.balakirsky@gtri.gatech.edu

Highlights

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 article 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 article 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.

***Detailed Response to Reviewers**

Response to reviewer 1:

1) Among the issues that caught my attention in the article is the fact that the author draws attention that a major limitation in manufacturing is the fact that work cells do not adapt even to small changes in the production environment and need to be reprogrammed offline. However, no reference is given for this assertion,
   A reference is provided for the fact that 95% of industrial robots lack sensing in their outer feedback loop. A statement that clarifies why lack of sensing leads to lack of flexibility to change has been added.
2) nor the presented example explicitly contemplates the case where late binding of part locations proposals could be applied to a real situation of environment change and reprogramming
   Additional text has been included in section 3.1 to address this. This augments the text that already presented an example in section 3.2.
3) In my opinion it is not clearly described the extensions that has been performed in order to remove the need to program the robot for new predicates and actions.
   I am not sure that I follow this comment. The entirety of section 4 (and section 4.4 in particular) describes additions to the ontology that encode predicates and actions as compositions of basic functions that make up a basis set of robotic vision and action primitives. This decomposition and representation in the ontology is the heart of this paper.
4) Another problem I encounter in the article is that it is very focused on itself. In the introduction the author does not enumerate or analyze other proposals that might exist and that could allow greater flexibility in the scheduling of work cells.
   Additional material is added to the introduction that speaks to several similar projects. This was also used to provide more recent references.
5) I also believe that the bibliography presented is quite limited. Reference [1] is a youtube video, which is quite unusual. Moreover, from the 9 references submitted, 30% are the author's own work and these are the only that is recent (> 2010).
   More recent references are now included. I have justified my accuracy claim by a more “reputable” reference and have even added an additional YouTube video. The YouTube videos are in place to show that the vendors’ claims of accuracy can be easily replicated by many different individuals.

Response to reviewer 2:

HIGHLIGHTS - A domain is not listed with which this research is being conducted. From the title and paper details, it appears that the domain is manufacturing. This should be explicitly clear in the highlights.
Not sure that I understand “highlights”. Manufacturing is one of the keywords.

PUNCTUATION - There are several instances throughout the document where punctuation is lacking (e.g. Line 3 - use a comma after 'However')
Fixed line 3. More specific instances would help to fix any other locations.

GRAMMAR
There are numerous uses of the phrases 'be able to' and 'in order to.' In most instances, these can be removed.
Only one instance of “be able to”, this was removed. Removed 4 of 5 occurrences of “in order to”.

Likewise, a sentence should never end with a prepositional phrase (e.g. Line 5)
Line 5 was fixed. More specific instances would help to fix any other locations.
Line 227 (working backwards from 230 on page 15) - Is 'lass' supposed to be 'class?'
Yes, this has been fixed.

ADD TECHNICAL CLARITY
Line 7 - What is an 'outer feedback loop.' This should either be defined or referenced.
Line 343-344 - Please elaborate on the statement of 'This work has been performed in simulation.' What were the findings? Will the simulation be used in further efforts? What did you learn from the simulation exercise?
The simulation framework was elaborated upon. Simple findings are listed, but detailed findings are beyond the scope of this paper.
Overall - This work appears to hold much promise, yet greater clarity could be provided to the manufacturing example to make it easier for the reader to understand the approach.
Various text has been added in order to attempt to clarify the manuscript.

ADD REFERENCES
Line 11 - A statement is made about just-in-time manufacturing and small batch processing that should be supported with at least 1 reference
By definition, a small batch is a limited quantity of a particular product. By observation, changing to a different product would require different robot programming. I am not able to locate a reference that proves this statement. If one is available, I would be happy to add it.
Line 20 - An IEEE WG is noting as taking the first steps to create a knowledge repository. At minimum, a website should be referenced indicating the specifics of these first steps and their overall plans.
Added reference.
Line 24-25 - An industrial subgroup is mentioned to have applied the infrastructure to a sample kit building system. This application, including any findings, should be referenced.
Overall - The paper only lists 9 references…that seems a bit light.
Several additional references have been added.

FIGURE REVISIONS
Figure 7 - 'place-kitTray' appears redundant…it shows up twice.

Fixed

Figure 8 - Recommend presenting this code in Pseudo-code. It would make the this code much easier to understand and follow

This is standard PDDL representation and would lose meaning if presented in pseudo-code.

Ontology Based Action Planning and Verification for Agile Manufacturing

Stephen Balakirsky

Georgia Tech Research Institute, Atlanta, GA 30332, USA

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 article 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 article 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.

Keywords: `knowledge driven system, adaptive planning, manufacturing, ontology, robotics, Planning Domain Definition Language`

*Corresponding author
Email address: `stephen.balakirsky@gtri.gatech.edu` (Stephen Balakirsky)

1. Introduction

Many of today's robotic arms are capable of obtaining sub-millimeter accuracy and repeatability. Robots such as the Fanuc LR Mate 200iD claim $\pm$ 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 requires changes in the manufacturing process on a batch-by-batch or 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 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 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. Figure 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

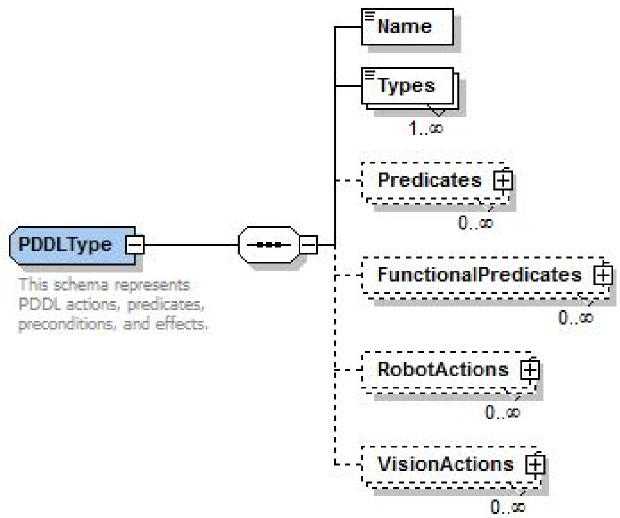


Figure 1: Description of the PDDLType class that is designed as an augmented PDDL description language. It contains all of the information necessary for interacting with the robot cell's robot controller and vision system.

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 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 *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

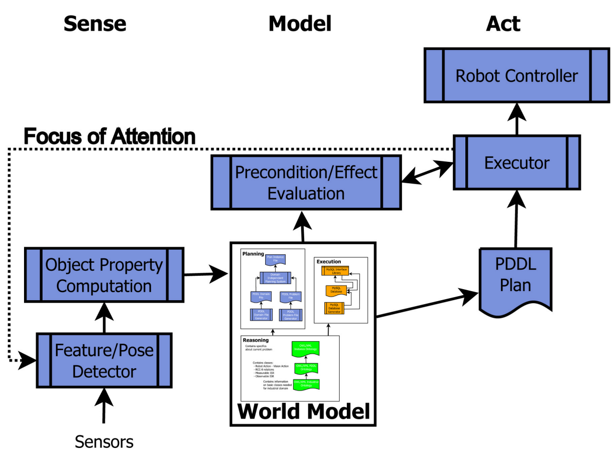


Figure 2: Major components that make up the Sense–Model–Act paradigm of the kitting station.

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 Figure 2, 4D/RCS follows a sense-model-act paradigm. A central feature of 4D/RCS is its world model. As shown in Figure 3, the world model for this system may be decomposed into the three parts of 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.

3.1. Reasoning

The reasoning portion of the world model is designed to contain all of the information needed to reason over and solve complex industrial problems. The knowledge is represented in an XML schema and is automatically translated into the Web Ontology Language (OWL) with tools described in Kramer et al. [13]. The ontology is structured in three parts. The first part of the ontology contains generic information and classes that are needed for the domain of kit building. This area of the ontology contains information on basic elements such as a "point" which is defined as a class that contains a name and a three-dimensional quantity, as well as complex types such as a "part", which is shown in Figure 4, and contains elements such as the part's location and a name that references a stock keeping unit. The stock keeping unit contains static information on classes of parts such as the part's shape, weight, and the end effector that should be used for grasping the part. This information is utilized to create parameters for the Planning World Model and the skeleton tables for the MySQL database of the Execution World Model.

Both static and dynamic information is represented in this ontology and is automatically transitioned into the Planning and Execution areas of the world model. During system operation, dynamic information is updated in the Execution World Model. More information on this portion of the ontology may be

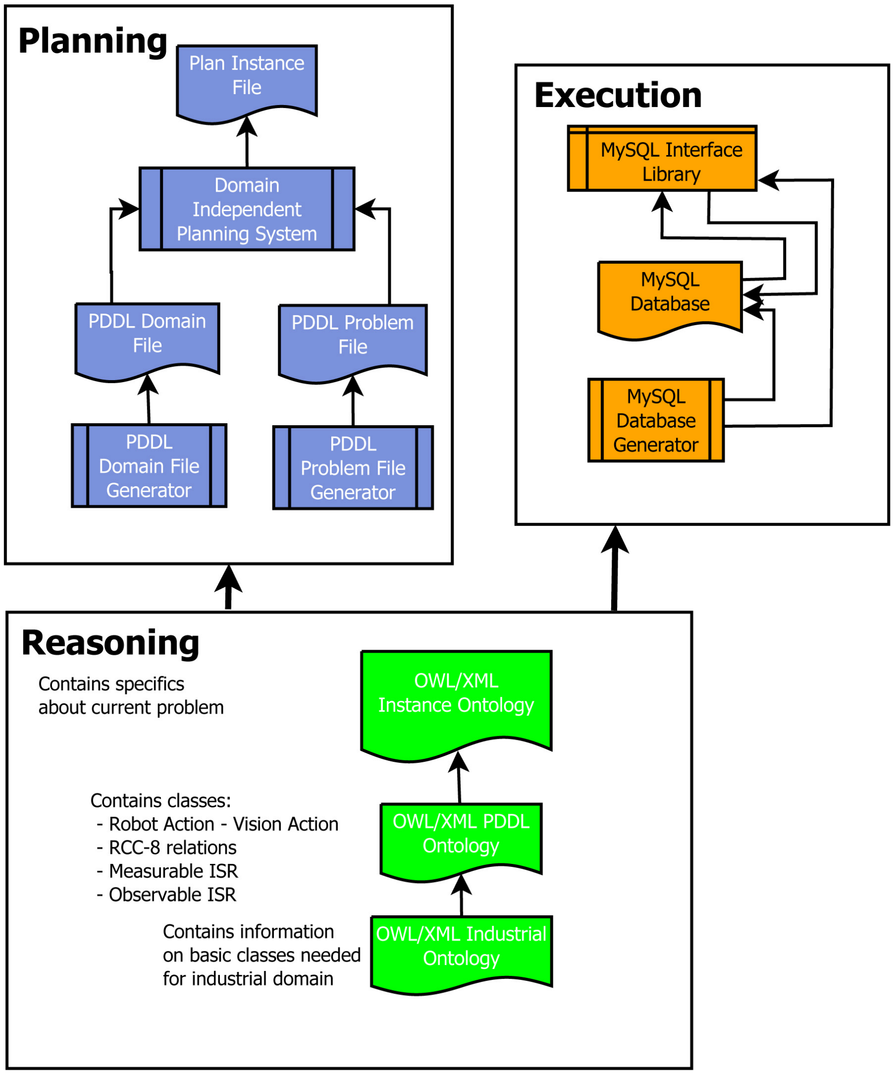


Figure 3: System World Model - The world model contains a Reasoning section that is based on an ontology shown in green, a Planning section that is based on a Planning Domain Definition Language (PDDL) specification shown in blue, and an Execution section that is based on a relational database (MySQL) shown in orange.

found in [14].

150 The second part of the ontology (known as PDDL ontology) contains the

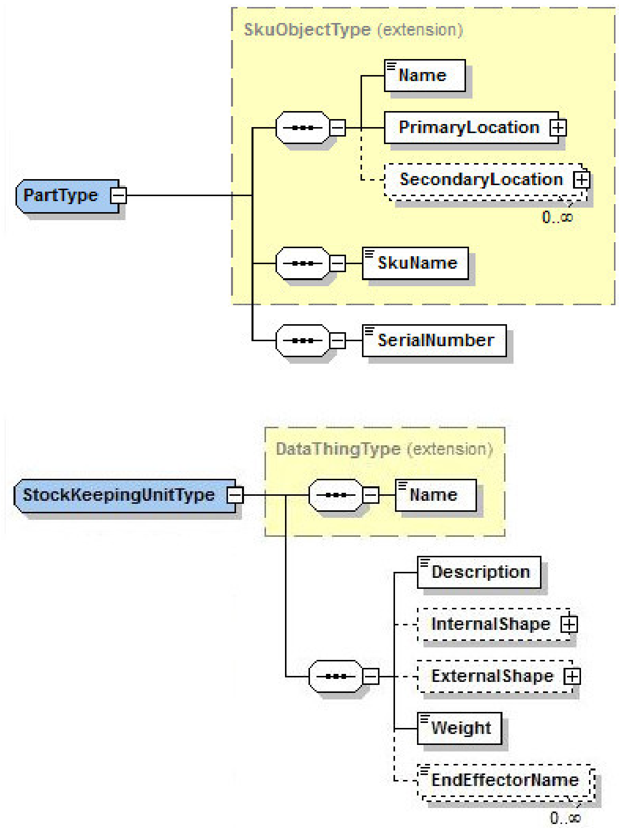


Figure 4: Description of the PartType class that is designed to contain both static and dynamic information about particular parts and the StockKeepingUnitType class that contains static information about classes of parts.

high-level concept of an action and all of the concepts that are required to support an action. Figure 5 depicts the template for the action representation. In addition, this portion of the ontology contains information on the set of functions that are hard-coded onto the robot and vision systems, and templates for how these functions may be composed into higher-level activities. More information on this action composition may be found in Section 4.

The third part of the ontology contains specific instances needed for a particular domain. All of the necessary information for the automatic generation

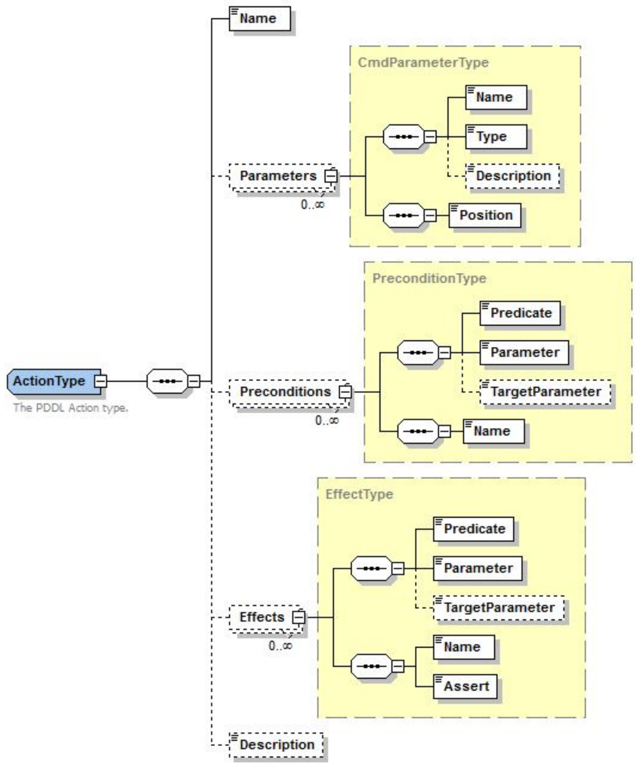


Figure 5: XML schema representation of a PDDL action. Contains representations of the action's parameters, preconditions, and effects.

of the PDDL domain file required by the planning system is contained in the combination of this instance ontology and the PDDL ontology.

One of the goals of this framework is to introduce additional agility into industrial processes. Therefore, partial information is accepted and even encouraged for this area of the ontology. For the example of a part shown in Figure 4, information on the SKU, grasp points (part of the ExternalShape or InternalShape), and name would be expected to be available at runtime. Information on the location of the part (PrimaryLocation) may not become valid until after a *VisionCommand* has been executed that has identified and located the particular part. This late-binding of a part's location allows the system to utilize pre-computed plans in conjunction with fixture-free part placement or parts located in an unstructured bin. The system is able to be tasked with retrieving an instance of a particular part type, with the actual instance being decided during system execution.

3.2. Planning Model

PDDL planners require a PDDL file-set that consists of two files that specify the domain and the problem. From these files, the planning system creates an additional static plan file. Both the domain and problem file are able to be auto-generated from the reasoning section of the world model.

The generated static plan file contains a sequence of actions that will transition the system from the initial state to the goal state. To maintain flexibility, it is desired that detailed information that is subject to change should be "late-bound" to the plan. In other words, specific information is acquired directly before that information needs to be used by the system. This allows for last minute changes in this information. For example, the location of a kit tray on a work table may be different from run to run. However, one would like to use the same planning sequence for constructing the kit independent of the tray's exact position.

To compensate for this lack of exact knowledge, the plans that are generated by the PDDL planning system contain only high-level actions. A representation of this plan may be stored in the ontology for future use.

3.3. Execution Model

The execution world model is also built automatically from the ontology. This world model consists of a MySQL database and C++ interfaces that provide for easy access to the data. The table skeletons are generated from the industrial ontology, and the tables are initially populated with information from the instance ontology. During plan execution, the Executor guides the sensor processing system in updating the information in this section of the world model. All of the data structures encoded in the ontology are included in this representation.

3.4. Executor

The overall framework is coordinated by a module known as the executor which receives its input from the domain independent planning system. Any one of a number of freely available open source PDDL planners such as the forward-chaining partial-order planning system from Coles et al. [15] may be used in conjunction with this work. The planner may compute plans *a priori* that are stored in a plan library for future retrieval, or may compute new plans as conditions change that invalidate the currently executing plan. The output of the PDDL planner is shown as the driving input to the executor in the lower right-side of Figure 2.

```
Data: kitToBuild
Result: reports success or failure
retrieve instance PDDLInstance to construct kit kitToBuild;
for each action A in PDDLInstance do
    for each precondition P of action A do
        if PredicateEvaluation(P) = false then
            report failure;
        end
    end
    ExecuteAction(actionA);
    for each effect E of action A do
        if PredicateEvaluation(E) == false then
            report failure;
        end
    end
    report action success;
end
report plan success;
```

Figure 6: BuildKit – Sequences the actions necessary to build a kit.

To construct a kit, the executor steps through each action in the precomputed PDDL plan. The overall process, known as BuildKit is described in Figure 6.

This process begins by retrieving a planning instance that has been precomputed to solve the construction of the requested kit (Line 1 of Figure 6). This PDDL plan is an ordered sequence of actions with each action containing types from the ontology as parameters. These types represent generic classes and are not yet grounded in specific instances that exist in the world. The PDDL actions may be broadly categorized into actions that are designed to ground objects to specific instances (Vision Actions) and actions that are designed to manipulate grounded objects (Robot Actions).

Vision Actions	Robot Actions
• look-for-endEffector-holder	• attach-endEffector
• look-for-endEffector	• detach-endEffector
• look-for-kitTray-holder	• init-robot
• look-for-kitTray	• take-kitTray
• look-for-workTable	• place-kitTray
• look-for-part	• take-part
• look-for-slot	• place-part

Figure 7: Family of PDDL actions necessary to build a kit.

The set of these actions for the kitting domain may be seen in Figure 7. For the domain of kitting, the Robot Actions are used to pick-up and place parts while the Vision Actions are used to located the parts and part destinations. The *for* loop beginning at Line 2 of Figure 6 steps through the execution of each of the actions from the PDDL plan. Before the action is executed, predicates in the form of preconditions are examined to assure that the action to be attempted is valid. If any of the action's preconditions are not able to be validated, a failure is reported; otherwise, the action is approved for execution. Once the action has been executed, additional predicates in the form of effects are examined to determine the success of the action. If any of the action's effects are not able to be validated, a failure is reported. If the action was successful, the loop will begin again with the next PDDL action. Once all of the actions have been successfully executed, plan success will be reported (line 16 of figure 6).

All of the steps of the BuildKit algorithm can be hard-coded into a robotic system as was reported in Balakirsky et al. [10]. This creates a system that is flexible and agile in terms of the type of kit that is being built, and the placement of parts, trays, and kits. However, if a procedural change is required that necessitates a new action, the system must be taken down and reprogrammed. For example, if a new "inspect-part" operation was desired, this action would need to be programmed into the system's vocabulary. The remainder of this article concentrates on techniques that allow more of this information to be stored in the system's ontology and thus allow even greater flexibility in what the system is able to accomplish without changes in programming.

4. Knowledge Representation

Two typical PDDL commands of *place-part* and *look-for-slot* are shown in Figure 8. The *place-part* command is used to put the currently held object into the "slot" of the kit under construction that was found with the *look-for-slot* command. If we assume that the action *place-part* is being executed, then in line 4 of the BuildKit algorithm each of the predicates from the precondition section of the action will be verified. As previously mentioned, the evaluation of these predicates could be hard-coded into the system. However, is there a more flexible way that this could be accomplished?

4.1. Predicate Representation

If one examines each of the predicates that must be evaluated, it may be noted that the predicates could be composed of a combination of simpler ex-

```
(:action place-part
    :parameters(
        ?robot - Robot
        ?stockkeepingunit - StockKeepingUnit
        ?endeffector - EndEffector
        ?kit - Kit)
    :precondition(and
        (endEffector-has-heldObject ?endeffector ?stockkeepingunit)
        (robot-has-endEffector ?robot ?endeffector)
        (equal-to (slot-found-flag) 1)
        (kit-has-slot ?kit))
    :effect(and
        (not(endEffector-has-heldObject ?endeffector))
        (set-value (slot-found-flag) 0)))
(:action look-for-slot
    :parameters(
        ?robot - Robot
        ?stockkeepingunit - StockKeepingUnit
        ?kit - Kit)
    :precondition(and
        (equal-to (slot-found-flag) 1))
    :effect(and
        (set-value (slot-found-flag) 0)))
```

Figure 8: Actions

pressions. For example, the predicate expression:

$$\textit{endEffector-has-heldObject}(\textit{endeffector}, \textit{stockkeepingunit}), \tag{1}$$

is designed to verify that the robot's end-effector is holding the correct class of part. It may be decomposed into the compound expression:

$$\textbf{matchSKU}(\textbf{In-Contact-With}(\textit{endeffector}), \textit{stockkeepingunit}) \qquad (2)$$

In this case, *endeffector* is the instance of the class end effector that is expected to be attached to the robot and *stockkeepingunit* is the instance of the stock keeping unit that belongs to the part that is expected to be held by the robot. The expression **In-Contact-With** will determine what class of object is being held by the effector, while the expression **matchSKU** will determine if this represents the expected class.

The expressions **In-Contact-With** and **matchSKU** are known as Intermediate State Relations (ISR) because the relate complex elements of the system's state to easily measurable or observable phenomenon. When called with a single parameter, the measurable ISR of **In-Contact-With**(*endeffector*) will return a list of objects that are in contact with the end effector. This list will be passed as parameters to the observable ISR of **matchSKU**. Since this relation is expecting exactly two parameters, an end effector touching zero or more than one object will result in an error. If a single object is being held, it will be passed to **matchSKU** where a simple observation may be made to see if the object's SKU matches the one provided in the second parameter.

4.1.1. RCC-8 Relation

All of the measurable intermediate state relations may be further decomposed into expressions that may be easily computed from knowledge of the six degree-of-freedom pose of the objects. These base expressions are know as Region Connected Calculus (RCC-8) relations and were originally developed by Wolter and Zakharyaschev [16] as an approach for representing the relationship between two regions in Euclidean or topological space. RCC-8 consists of eight possible relations (hence RCC-8) that include measurable region-to-region relationships such as Tangential Proper Part (TPP) and Externally Connected (EC).

To represent these relations in all three dimensions for industrial domains, RCC-8 has been extended to a three-dimensional space by applying it along all three planes (x-y, x-z, y-z) and by including cardinal direction relations "+" and "-". In our example of Equation 2, **In-Contact-With**(*endeffector*) may be expressed in RCC-8 relations as:

$$\textbf{In-Contact-With}(\mathit{obj1}, \mathit{obj2}) \rightarrow$$
$$\texttt{x-EC}(\mathit{obj1}, \mathit{obj2}) \vee \texttt{y-EC}(\mathit{obj1}, \mathit{obj2}) \vee \texttt{z-EC}(\mathit{obj1}, \mathit{obj2})$$

Where *EC* stands for Externally Connected, obj1 is the end effector, and obj2 is cycled through all of the detected objects in the work cell.

4.2. Observable Relations

Work is still being performed to relate all of the simple observations to observable relations. In the case of the **matchSKU** observation, a *model-match* or *view-sku* operation could be performed. More on this topic is presented in Section 5.

4.3. Action Execution

Once the precondition predicates have been validated, line 8 of the BuildKit algorithm in Figure 6 shows that the action should be executed. Once again it is possible to decompose the complex actions into a much simpler form. In this case, two separate basis sets exist with one designed for the decomposition of Robot Actions and the other for the decomposition of Vision Actions.

4.3.1. Canonical Robot Command Language

The National Institute of Standards and Technology (NIST) has developed a robot language known as the Canonical Robotic Command Language (CRCL) that is designed to provide a basis set of operations for robotic arms with industrial use cases [14]. This language contains 22 command elements that may be sequenced to perform any of the PDDL Robot Actions that have been defined for this domain. Continuing our example from above, Figure 9 displays

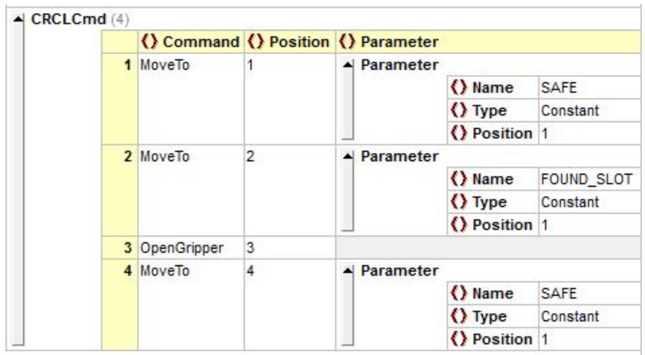


Figure 9: The *place-part* action may be decomposed into a sequence of four CRCL commands.

the decomposition of the *place-part* action. As may be seen in the figure, the command decomposes into several movements and a gripper command. The movements to a "safe" location are not strictly necessary, but are included to assure that the arm's motion begins and ends from a known safe position. This position is a predefined constant in the system. *FOUND_SLOT* is another system constant that represents a position buffer in the system. This buffer has previously been filled with the location of the actual slot that is the destination of the part with the PDDL action *look-for-slot*.

Once the command execution is complete, the predicates that make up the effects are examined in the same manner as the previously examined precondition predicates. If any of the effects are deemed to have not occurred, then an error will be generated.

4.4. Instance Representation

As discussed in the previous section, all of the PDDL predicates and actions may be decomposed into a small set of primitives that includes Robot Actions, Vision Actions, RCC-8 relations, and Observables. If this set of primitives is programmed into the robot cell, then the set of behaviors that the cell is capable of may be altered through the creation of new PDDL actions and predicates rather than reprogramming the cell. In addition, these PDDL actions and predicates are independent of the actual cell's implementation and product vendors.

As long as the cell supports the full set of primitives, it is capable of carrying out the operations without programming.

To translate PDDL predicates and actions into robot cell commands, the executor needs to have an understanding of how the predicates and actions are composed. Since this information is designed to not be hard-coded on the robot, it must be accessible to the running system. In addition, since the actions and predicates are designed to be expanded upon as new activities are added to the robot's vocabulary, the information needs to be accessible in a human friendly form. To meet all of these demands, this information is encoded in

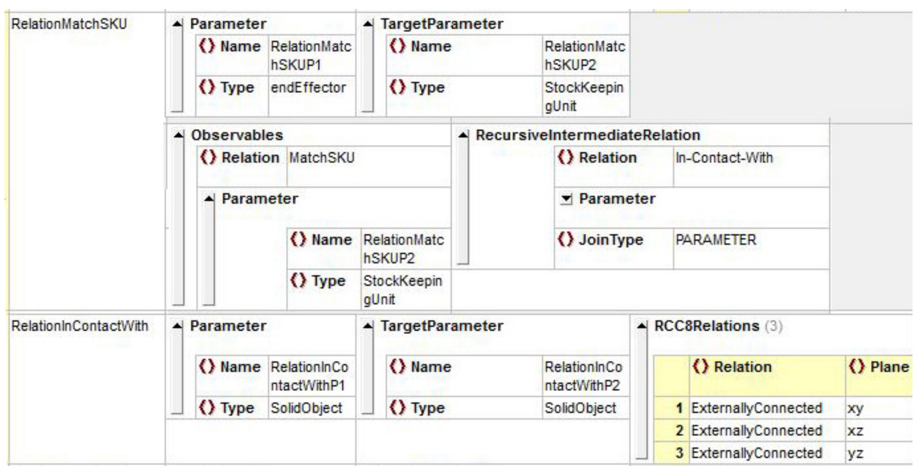


Figure 10: Intermediate State Relation RelationMatchSKU. This is the implementation of Equation 2.

the Instance Ontology and is automatically entered into the Execution Model's MySQL database.

The primitives themselves are implemented as simpleTypes in the schema. In this case, the XML simpleType specifies an enumerated list of terms that are within the robot and vision system's vocabulary. All of the ISRs, predicates, and actions are composed of these elements, and these elements are the only part of the system that is hard-coded onto the robot.

4.4.1. Intermediate State Relation

The structure of the RelationMatchSKU ISR required by Equation 2 is shown in the XML representation depicted in Figure 10. Recall from Equation 2 that this ISR is a recursive combination of the Robot ISR **In-Contact-With** and the Vision ISR **MatchSKU**. This is shown in the XML in that the *RecursiveIntermediateRelation* is instantiated as **In-Contact-With** with a single parameter of the end effector and the *JoinType* of *PARAMETER*. This will cause a determination of every object that is in contact with the effector. The determination of which objects are in contact with the effector is made through the application of RCC-8 relations as shown in the bottom part of Figure 10. In this case, three externally connected relations must be evaluated for each object.The list of objects in contact will get passed back as parameters to the *MatchSKU* Vision ISR. If zero or greater than one objects are passed to *MatchSKU*, an error will be generated. If and only if one object is passed as a parameter, that object's SKU will be compared against the target SKU. Thus, from a combination of primitives we are able to determine if the object being held by the end effector matches the SKU of the expected object class. This in turn will deliver the truth value of the predicate *endEffector-has-heldObject()* depicted in Equation 1.

4.4.2. Actions

CRCLActionTypes and *VisionActionTypes* are included in the xml schema and are extensions of the *ActionType* depicted in Figure 5. These actions add the list of CRCL commands or observations that are required for the execution of the action. A sample of the CRCL commands for the *place-part* action are shown in Figure 9. The encoding of all of the actions is performed in the instance ontology.

5. Conclusions and Future Work

The framework described in this paper has been applied in simulation to the domain of kit building, which is a simple, but practically useful manufacturing/assembly domain. The simulation features a hardware-in-the-loop design where the actual planning systems are running and communicating through CRCL commands to simulated hardware. Through this system's use, we have been able to demonstrate agility in both kit construction through late binding of part locations, and in the addition of additional actions and predicates to the system's vocabulary.

While the basis set of actions is well understood (the NIST CRCL), the vision action basis set is still not fully defined. Work is underway to complete this definition, and the system will be modified to utilize the complete set when it becomes available. In addition, work is currently underway to construct an implementation of the system on real robotic hardware. It is believed that once the basis set of operations has been programmed on the robot, the rest of the system will port without any code modifications.

Extensions are also being investigated that will expand this work to the realm of general assembly. We hope to apply this knowledge based framework to simple assembly tasks (growing towards more complex tasks) on a real robot workcell in the near future.

[1] F. Robot, Fanuc robot lr mate 200id, in: Data Sheet, 2013.

[2] Control, M. Robotics Lab at the ETS, Measuring the absolute accuracy of an abb irb 1600 industrial robot, http://www.youtube.com/watch?v=d3fCkS5xFlg.

[3] O. Hnninen, Repeatability test fanuc robot, in: https://www.youtube.com/watch?v=8DMT5Aj_jjc, 2014.

[4] S. Department, World robotics industrial robotics, Tech. rep., International Federation of Robotics (2012).

[5] J. Huckaby, H. I. Christensen, A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics, in: Workshops at 26th AAAI Conference on Artificial Intelligence, 2012.

[6] J. Pfrommer, M. Schleipen, J. Beyerer, Pprs: Production skills and their relation to product, process, and resource, in: Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on, IEEE, 2013, pp. 1–4.

[7] R. Patel, M. Hedelind, P. Lozan-Villegas, Enabling robots in small-part assembly lines: The" rosetta approach"-an industrial perspective, in: Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on, VDE, 2012, pp. 1–5.

[8] M. Stenmark, J. Malec, K. Nilsson, A. Robertsson, On distributed knowledge bases for small-batch assembly, in: Cloud Robotics Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013.

[9] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, E. Miguelanez, An ieee standard ontology for robotics and automation, in: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, 2012, pp. 1337–1342.

[10] S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, S. Gupta, Knowledge driven robotics for kitting applications, Robotics and Autonomous Systems (2013) – doi:http://dx.doi.org/10.1016/j.robot.2013.04.006. URL http://www.sciencedirect.com/science/article/pii/S0921889013000602

[11] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, Pddl–the planning domain definition language, Tech. Rep. CVC TR98-003/DCS TR-1165, Yale (1998).

[12] J. Albus, 4-d/rcs reference model architecture for unmanned ground vehicles, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2000, pp. 3260–3265.

[13] T. Kramer, B. Marks, C. Schlenoff, S. Balakirsky, Z. Kootbally, A. Pietromartire, Software tools for xml to owl translation, To be published 1.

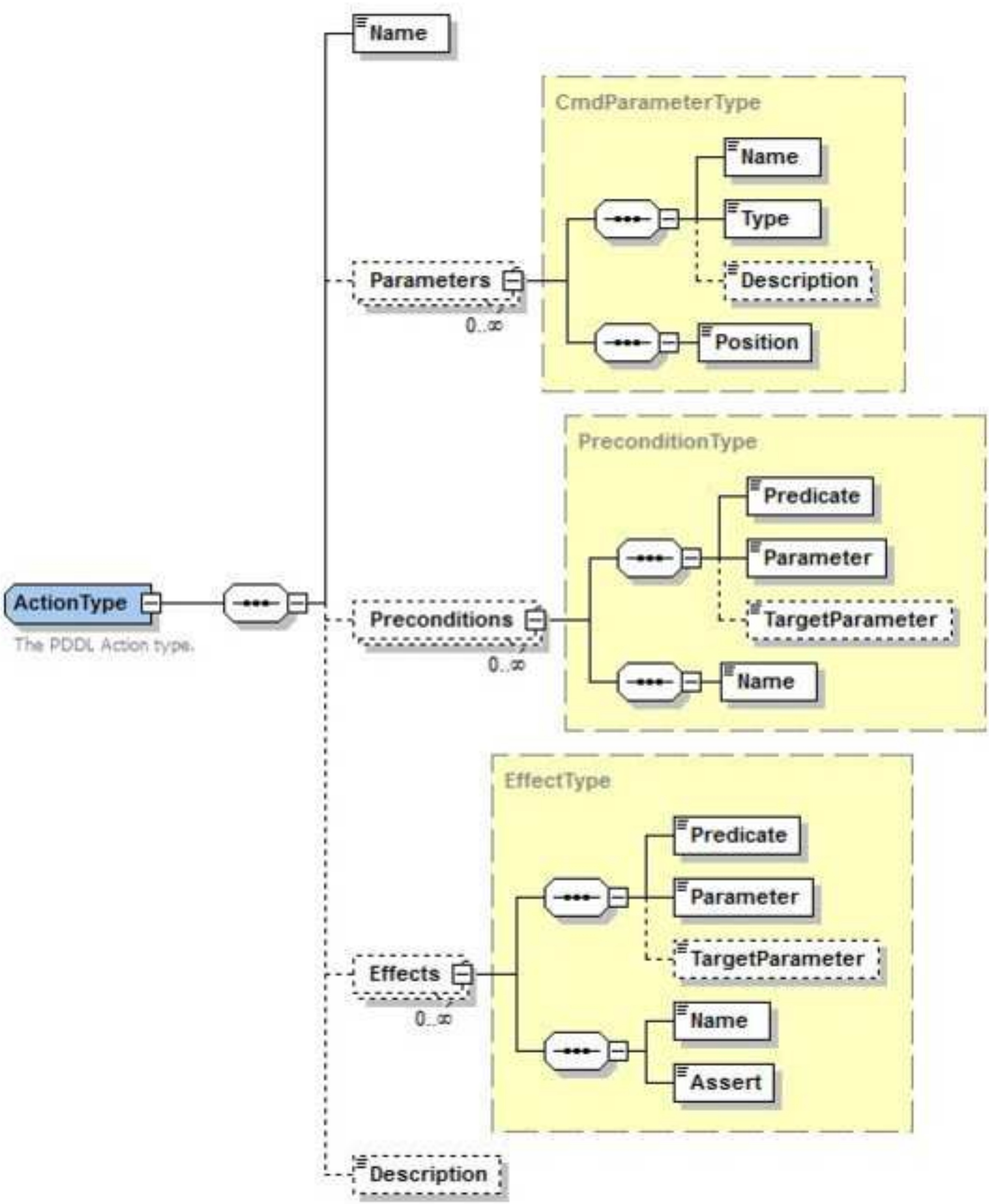
[14] S. Balakirsky, T. Kramer, Z. Kootbally, A. Pietromartire, Metrics and test methods for industrial kit building, NISTIR 7942, National Institute of Standards and Technology (NIST) (2012).

[15] A. J. Coles, A. Coles, M. Fox, D. Long, Forward-chaining partial-order planning, in: 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, AAAI 2010, Toronto, Ontario, Canada, 2010, pp. 42–49.

[16] F. Wolter, M. Zakharyaschev, Spatio-temporal representation and reasoning based on rcc-8, in: Proceedings of the 7th Conference on Principles of Knowledge Representation and Reasoning, KR2000, Morgan Kaufmann, 2000, pp. 3–14.

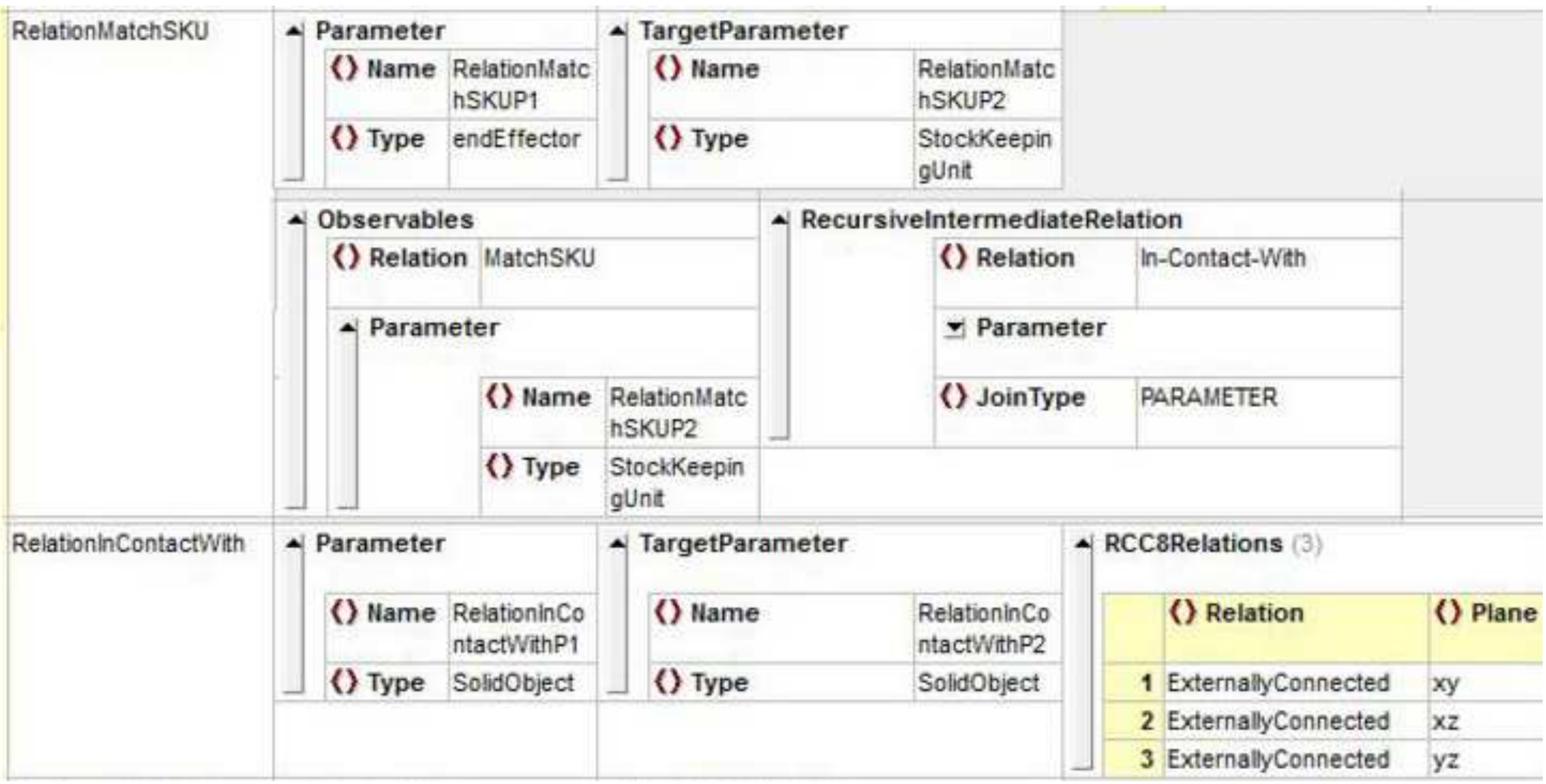
actiontype.jpg

Click here to download high resolution image



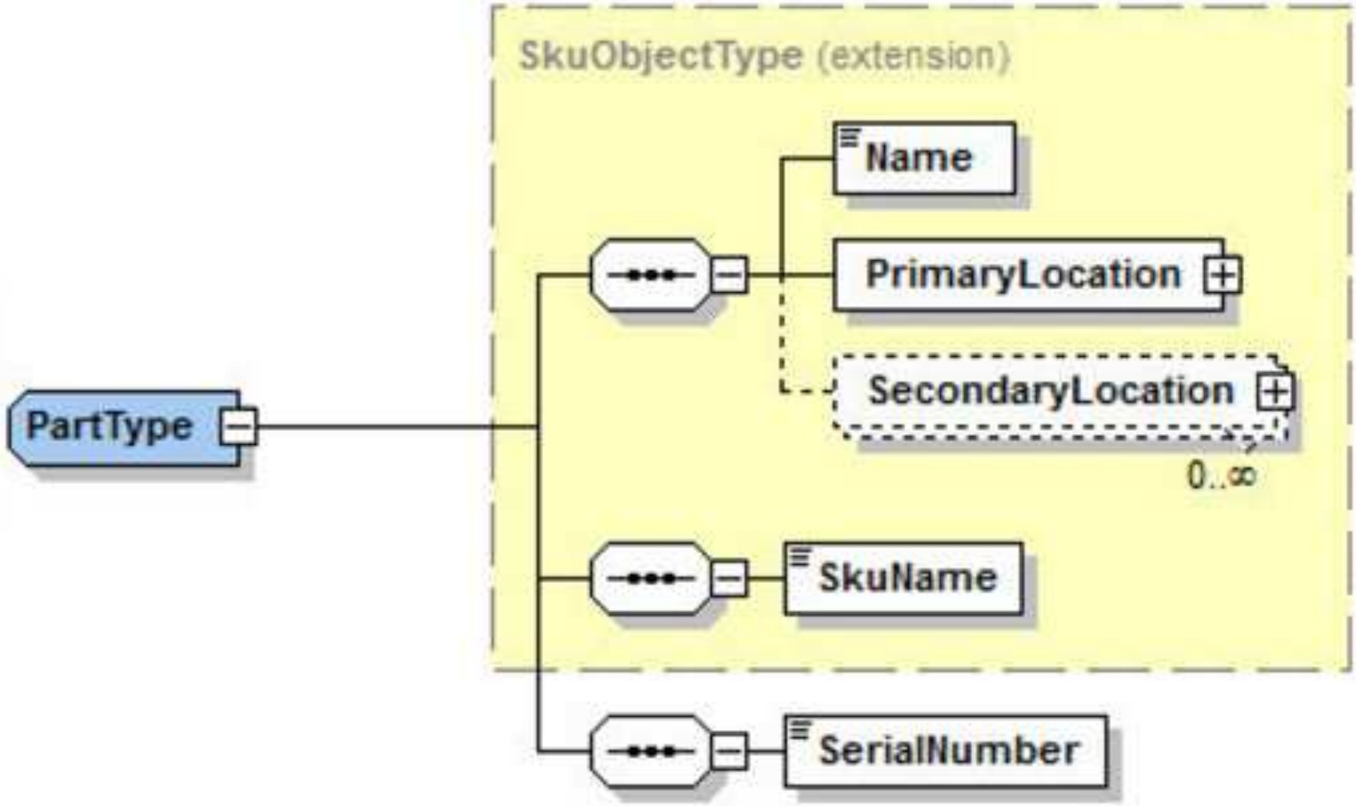
matchsku_isr.jpg

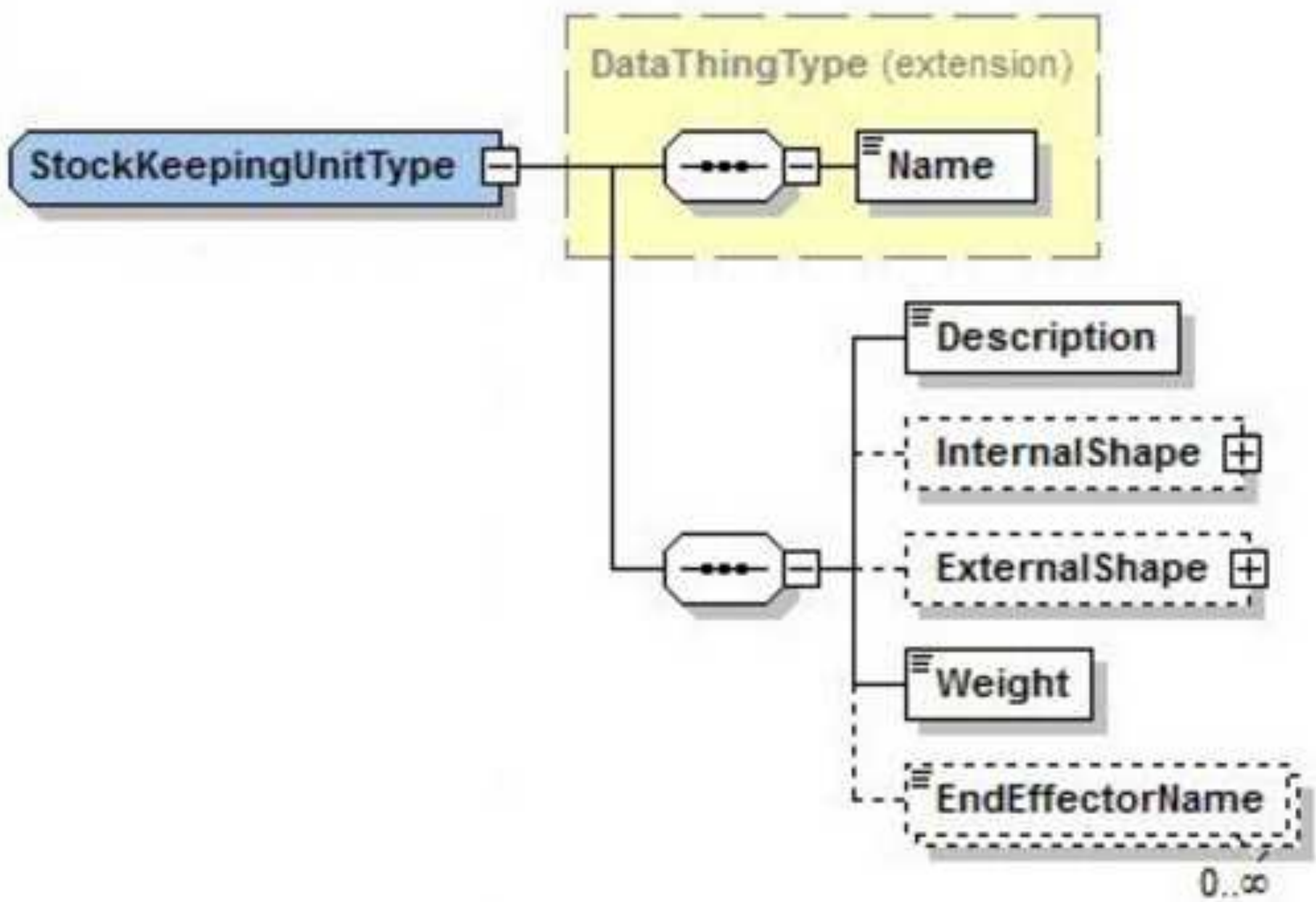
Click here to download high resolution image



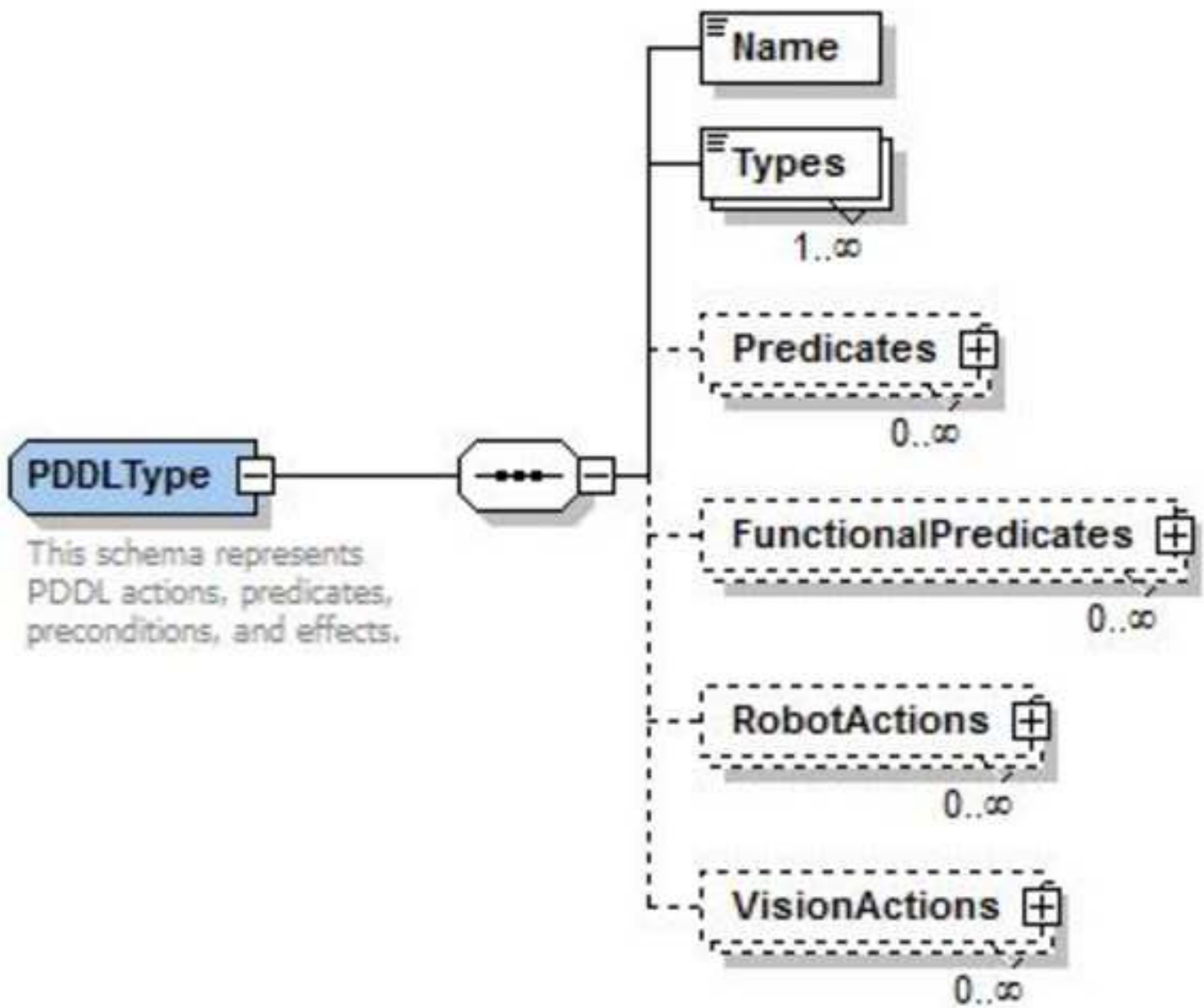
partskuv2.jpg

Click here to download high resolution image





pddltype.jpg
Click here to download high resolution image



placepartcrcl.jpg

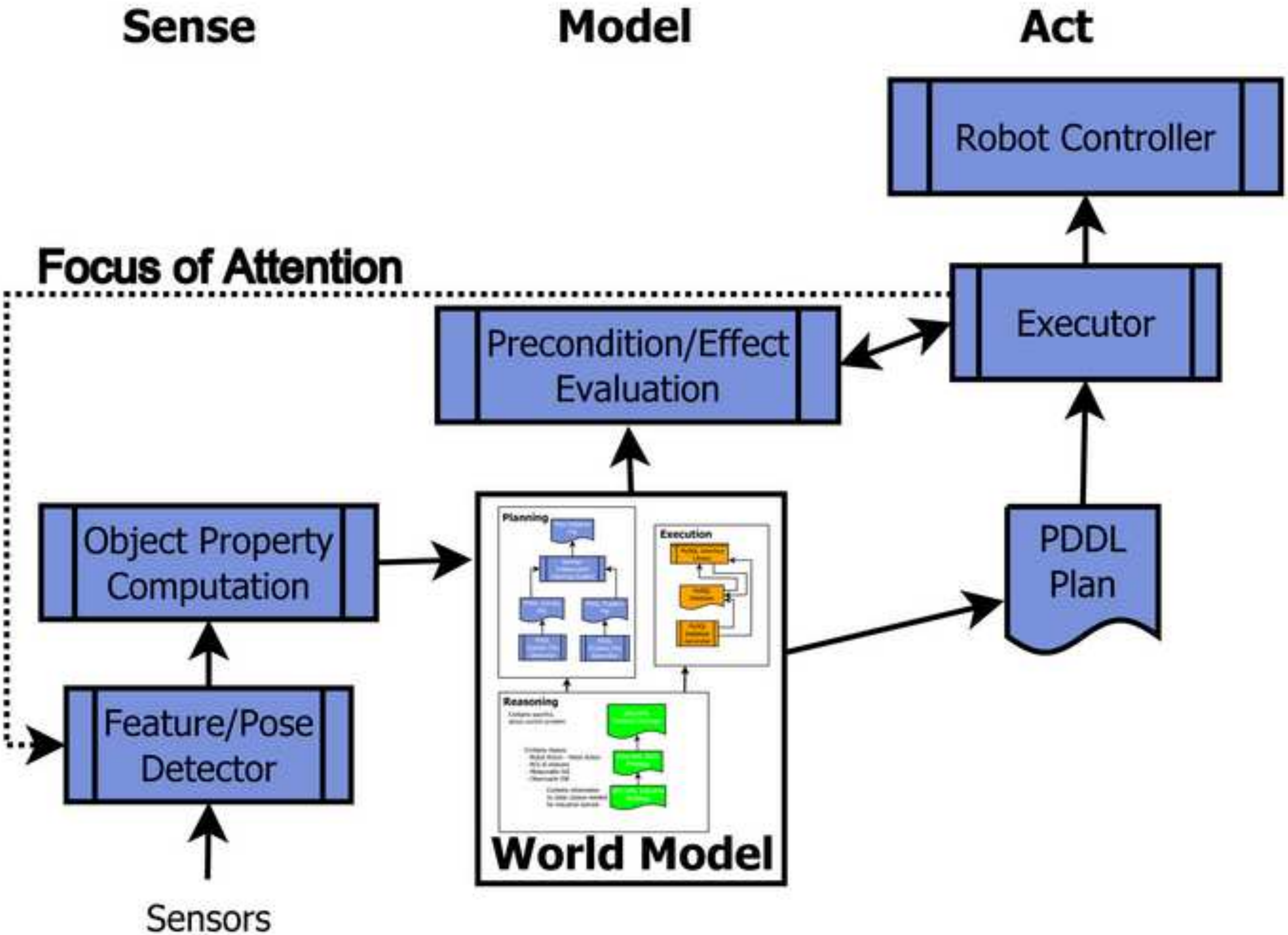
Click here to download high resolution image

CRCLCmd (4)

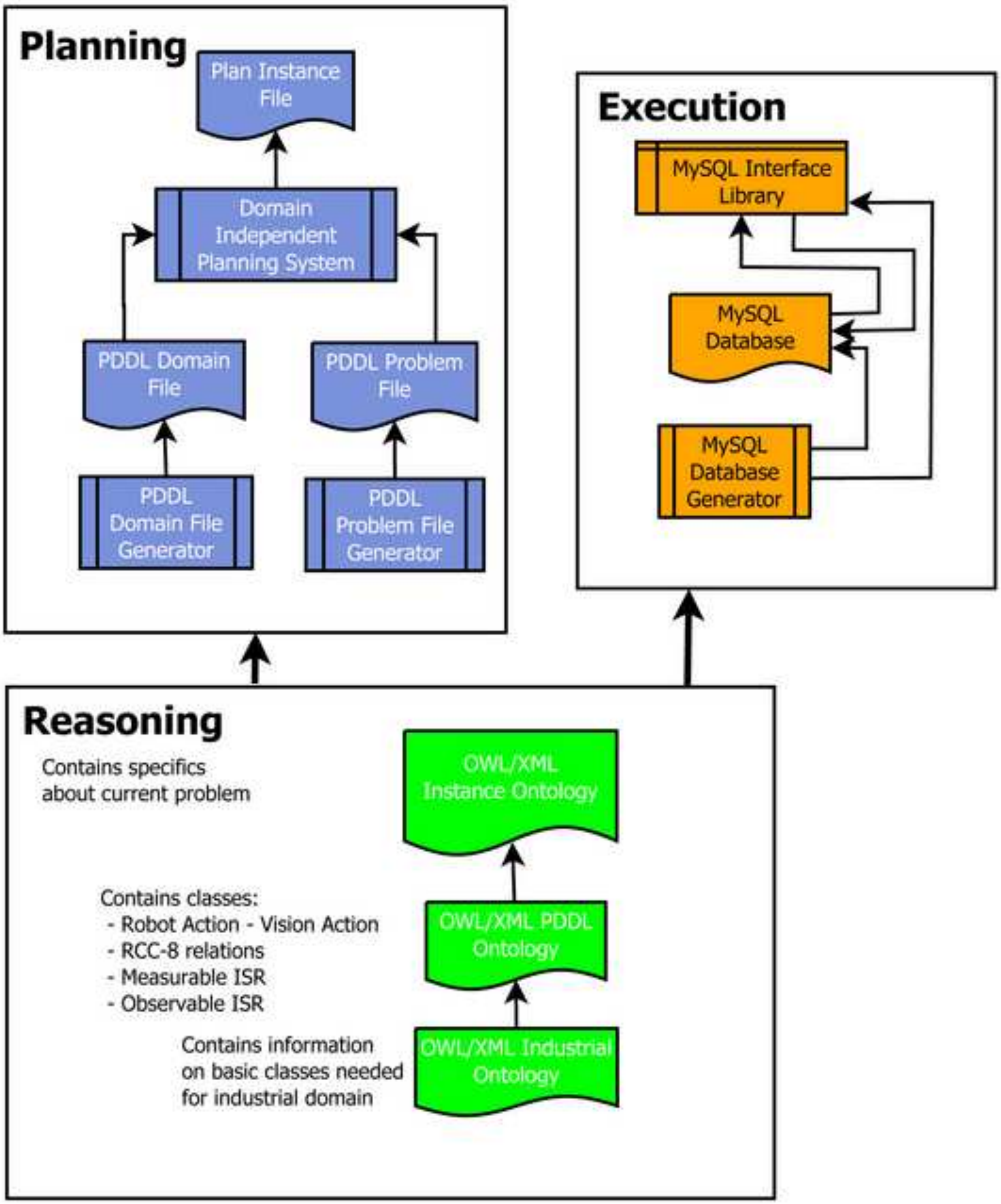
	Command	Position	Parameter
1	MoveTo	1	Parameter Name SAFE Type Constant Position 1
2	MoveTo	2	Parameter Name FOUND_SLOT Type Constant Position 1
3	OpenGripper	3	
4	MoveTo	4	Parameter Name SAFE Type Constant Position 1

sensemodelact.jpg

Click here to download high resolution image



worldmodel.jpg
Click here to download high resolution image



introduction.tex
Click here to download LaTeX Source Files: introduction.tex

pddl.tex

Click here to download LaTeX Source Files: pddl.tex

LaTeX Source Files

Click here to download LaTeX Source Files: systemOperation.tex

knowledgerepresentation.tex

Click here to download LaTeX Source Files: knowledgeRepresentation.tex

ontology.bib