

# The itSIMPLE tool for Modeling Planning Domains

Tiago Stegun Vaquero<sup>1</sup> Flavio Tonidandel<sup>2</sup> José Reinaldo Silva<sup>1</sup>

<sup>1</sup>Escola Politécnica – Universidade de São Paulo  
Design Lab. – PMR – Mechatronic and Mechanical Systems Department - São Paulo, Brazil

<sup>2</sup>Centro Universitário da FEI  
IAAA – Artificial Intelligence Applied in Automation Lab - São Bernardo do Campo, Brazil  
Email: tiago.vaquero@poli.usp.br; flaviot@fei.edu.br; reinaldo@usp.br

## Abstract

A graphical interface for the modeling of planning environments where an integrated tool permits the user to export the planning model to different representation languages such as PDDL or XML. The application uses an UML model to introduce a planning domain as a first step, followed by a step where a representation in Petri Nets – automatically translated from UML - is used to validate its static and dynamic behavior. A preliminary version of a software tool called itSIMPLE is presented which can manage the initial modeling on UML extended model and export the model to PDDL or Petri Nets using XML as internal language.

## Introduction

The knowledge engineering has received much attention in the planning community nowadays. One reason for that is the increased demand to create and validate more complex and real planning domains.

That is a suitable situation to introduce a tool that helps the knowledge acquisition, permits the validation of static and dynamic behaviors, and improves the portability between planning systems and real world domains. Some effort towards this kind of tools has appeared in the last years. One example is the GIPO (Simpson et al, 2001) software interface that works with an object-oriented approach and performs the analysis and validation of domains.

In this work we focus in the use of an integrated and portable tool that uses UML (Unified Modeling Language (D'Souza and Wills 1999)) as a modeling language for planning domain; the XML (Extended Markup Language) (Bray et al, 2004) acts as an internal language for compatibility with other representations, including PDDL (Fox and Long, 2003), and the Petri Nets (Murata 1989).

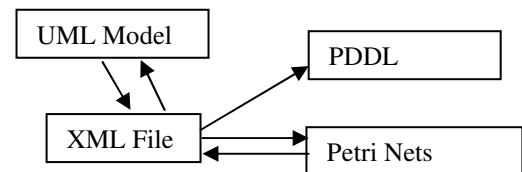
We aim to improve this integrated tool, called itSIMPLE (Integrated Tools Software Interface for Modeling Planning Environment), into a new environment that permits the static and dynamic validation of real domains

for planning. The main feature of itSIMPLE is to manage well-known modeling concepts and good practices that increase the portability of real world domains and facilitate its conversion to a planning domain.

Section 2 presents some details and support for the use of each particular tool in itSIMPLE software, as well as a brief explanation of how to use them. Section 3 shows a simple example and some screenshots of its interface. Finally, section 4 analyzes the results and goes to conclusion.

## The itSIMPLE software

Focusing on the portability among modeling languages used in industries and in planning environments, we propose an integrated tool called itSIMPLE that has the general structure depicted in Figure 1.



**Figure 1** – The structure of the integrated tools used by itSIMPLE

There are other tools to interface different representations for planning problems or in general. itSIMPLE intends to be a tool that inserts plan domains in the context of general problem solving, where the first steps are considered of great importance to the success of the project. Therefore the idea incorporated in the software tool is to have a disciplined process of elicitation, organization and analysis of requirements, before the choice of a planner. The change between representations are faced even for convenience (as it happens in the use of PDDL) or to perform a formal analysis, in Petri Nets for instance.

The proper way to activate a sequence of representations such as (UML, XML and PetriNets) is described below.

## Why UML?

We believe that most knowledge engineers in several application areas are somehow familiar with UML language. Besides, there are many planning applications described in UML models, like (Scheetz et al. 1999). This fact, besides the suitability of UML to make a first model (tracking requirement specifications) (Silva and Santos, 2004) turn it in a good choice.

In fact, if the planning community intends to build a bridge over the gap between real applications and planning simulation domains, it is important to have some flexibility in modeling languages used by both sides to model their applications. That is why we propose the UML language.

UML is an object-oriented language based on diagrams, such as class diagram, state chart diagram and object diagram. The objects and classes are linked by associations, aggregations and compositions. With all these features the planning domains can be represented and modeled.

The itSIMPLE permits to model planning environments in UML by defining a general structure composed by Agents, a domain Environment and a Planner. Agents change the arrangement of objects which compose the environment in order to go from an initial state to a goal state. The Planner just plans a sequence of actions to make the changes. These three elements are in Figure 2.

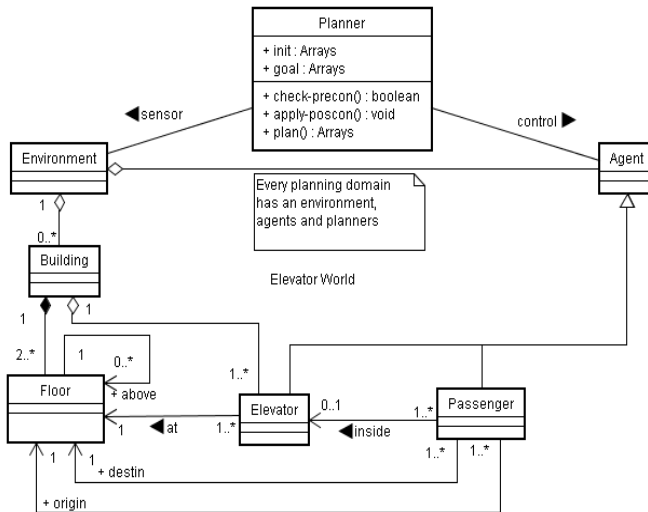


Figure 2 – Elevator domain modeled in UML

The class Planner controls all Agents and interacts with the Environment through sensors. Since Agents belong to the Environment, there is an aggregation association between Environment and Agent. The multiplicity 1 to 1..\* attached to this association means that a single Environment can have one or many Agents. The class Planner has two attributes by default: *init*, representing the initial state of the problem, and *goal* representing the goal state. It has also default methods called: *check-preconds* which checks an action pre-conditions; *apply-postcond* which applies

actions post-conditions, and *plan* which plans the sequence of actions to be applied in order to change states from initial state to the goal state.

It is also possible to define methods and global variables that control a plan quality. For example, in Depots domain the class Planner must have a new attribute called *fuel-cost* and a new method called *minimize(fuel-cost)*. After defining the global variable *fuel-cost* in the Depots domain, the post-condition of the action *drive* must change the value of this variable. A good example of these expressions would be: every time that the Truck is driven, using the action *drive*, the action must increase the variable *fuel-cost*. So the expression could be: *Planner.fuel-cost += 10*.

In a class diagram, the objects have attributes that correspond to nouns followed by possessive phrases, such as “the *capacity* of the plane” or “the *load limit* of a truck”. They can also correspond to some characteristics that represent the object state, such as “the block is *clear*” or “the truck is *empty*”.

Some attributes are related to associations. For instance, some cities are connected to each other by roads which distances are known. There is an association called, for example, “*road connected*” which has multiplicity 1 to 0..\* (i.e., one city can be road connect to none or many cities) and we have an attribute attached to it that could be called *distance*.

Agents can change the planning domain state by performing actions. Each agent can have a set of actions that it can perform in the domain. In the Graphical Specification, dynamic classes represent those objects which changes state during time. It is necessary to have one state chart diagram for each dynamic object.

Preconditions are specified in brackets [*pre*] and can compare attributes of the same class or some associations. The Effects are specified in /*pos* and they can group Proper effects (those that affect the Agent that performs the action), Direct Side Effects (those that affect other objects directly) and Indirect Side Effects (those that affect objects indirectly, as a side effect).

Each action transforms one state (S1) into another state (S2) in the diagram. In the classical view, attributes of S1 state compose pre-conditions and the delete list of the action, while attributes of S2 contribute, together with /*pos* specification, to compose the addition list of the action.

In UML, it is also possible to model invariants. Some attributes can keep their values constant or have their values derived from other attributes. UML can also represent association invariants, precondition invariants and invariants effects.

Finally, a planning problem can be modeled as a snapshot of the initial and goal states. A snapshot is a picture of a specific time.

## Why XML?

It could be argued that a new software interface for modeling domains in UML language is not necessary, since there are many other commercial tools that accomplish the same task. However, each one has its own file specification

for saving UML information, what would demand a specific translator to each new representation used to analyze the planning problem.

itSIMPLE software uses a XML specification to save the UML model in a file. This characteristic permits any internet browser or almost any developing computer language to read the file.

Following the principle to make our tool portable and regular as much as possible, XML files that are the output from the UML interface can be transformed directly to any other representation such as Petri Nets (using PNML library) or PDDL (using a proper prospective PDML library).

In XML files, everything is organized as a tree of diagrams, classes and attributes. There are some labels called ID that identify each node of the XML tree, what permits the specification of the associations and relations through these ID numbers.

### Why Petri Nets?

The validation of the static behavior of domains is very important. However, it is not enough to provide a complete domain analysis. Besides static behavior, there is the dynamic behavior of the domain.

Petri Nets, or Object-oriented Petri Nets, can provide a dynamic model of the planning environment in order to let the user to analyze dynamic behavior and also extract some important features about the use of certain heuristics. We also hope that with the Petri Nets model we can extract

enough information and patterns to decide which heuristic, or which planning technique, can be used in each domain.

In addition, the Petri Nets are also widely used in industry for modeling manufacturing process and it can be useful to approximate the manufacturing area to planning community.

Our software intends to provide a both way translation between Petri Nets and XML. It means that some Petri Nets models can be exported to XML file and, consequently, to the UML model.

Thus, a generic planning system, designed to read UML modeled domains could also be able to read Petri Nets models of a manufacturing or industrial processes, what could approximate general planning techniques to real world problems.

### Screenshots of the itSIMPLE

itSIMPLE software has a friendly interface with easy and intuitive commands and configuration.

Figure 3 shows itSIMPLE software modeling a Logistic domain in UML language. Figure 4 shows the same domain described in XML specification and Figure 5 shows its translation to PDDL language.

All screenshots are interconnected and can be accessed any time by a simple click on the bottom of the screen. In the future, features like domain validation and analyses using Petri Nets will be available.

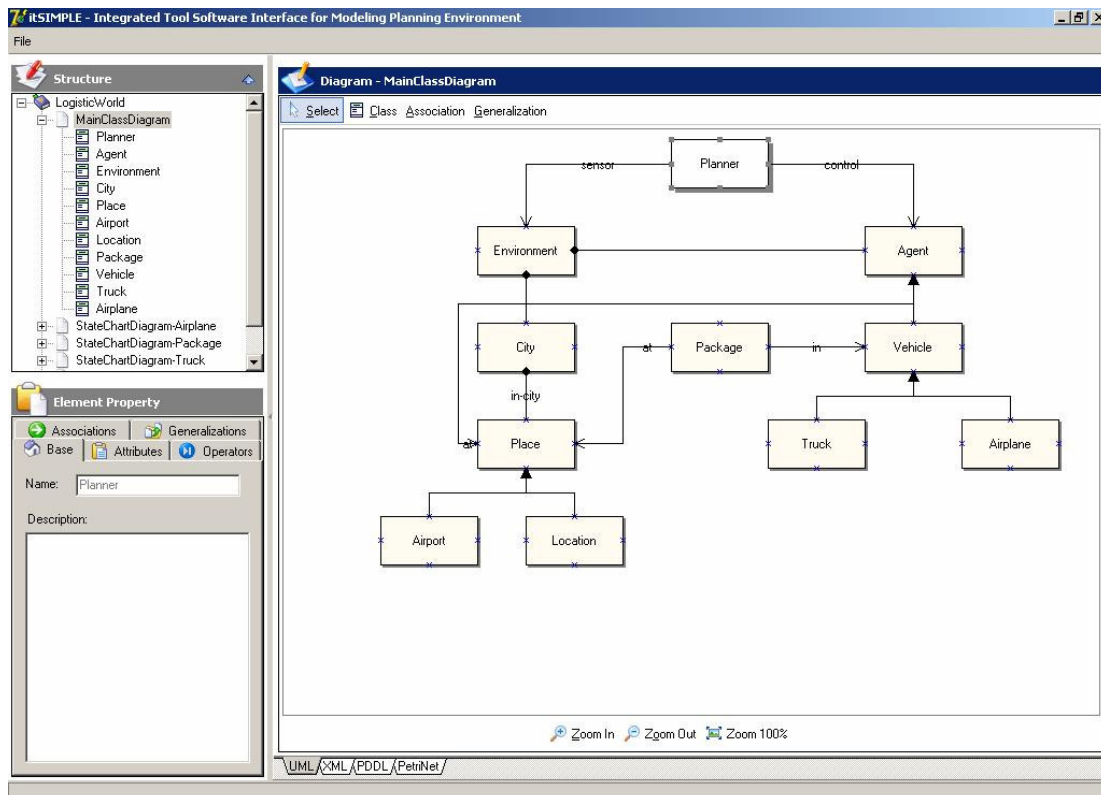


Figure 3 – Screenshot of a Planning UML model in itSIMPLE software

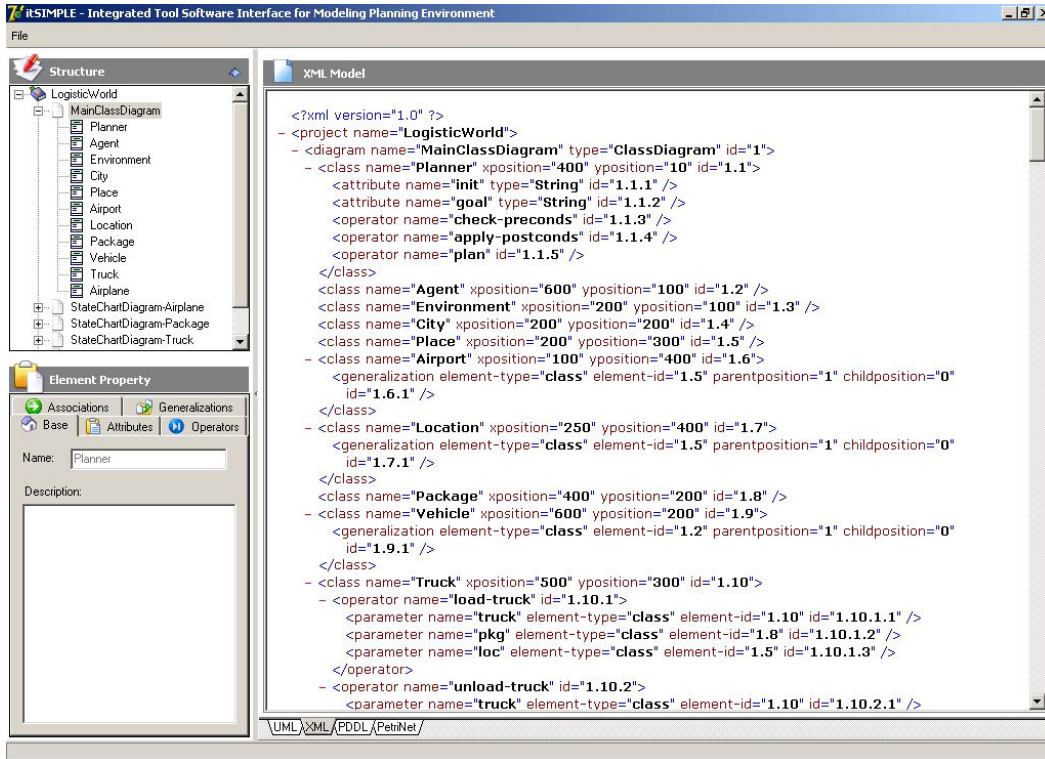


Figure 4 – Screenshot of XML file

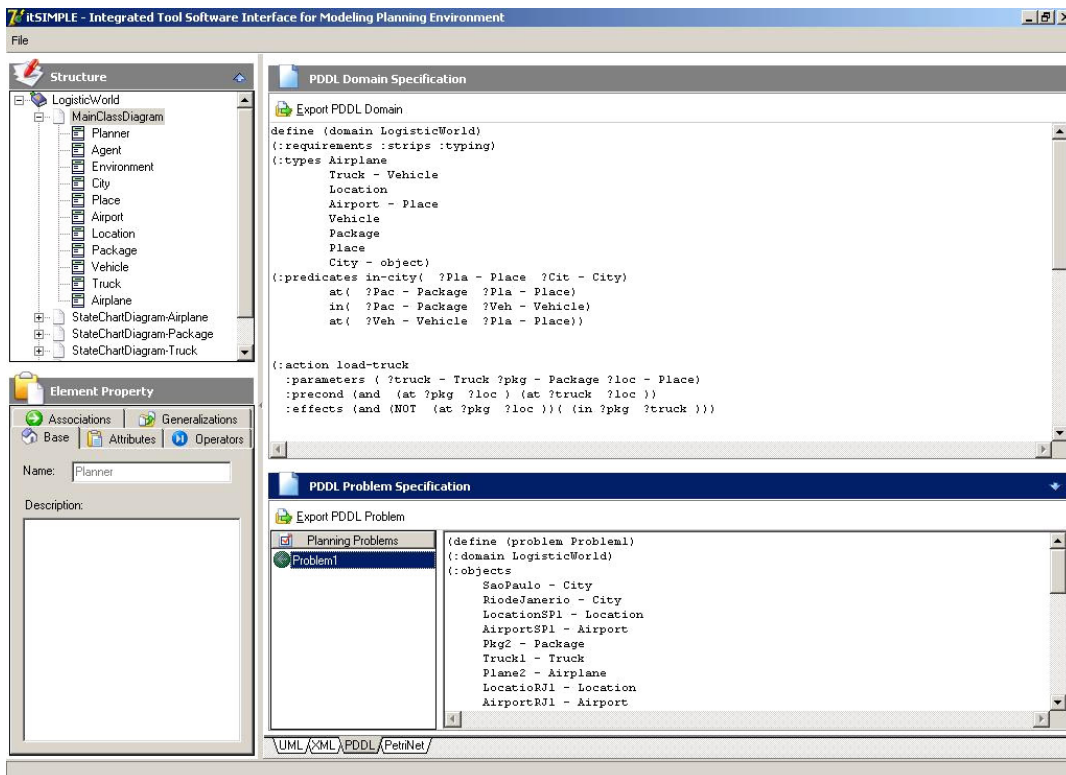


Figure 5 – Screenshot of the domain exported to PDDL

## Domain sample using itSIMPLE

This section presents a well known model of a classical planning domain, the Blocks World, using itSIMPLE. The preliminary blocks world model is constituted by UML diagrams. The first step follows insights from Naked Object approach (Pawson and Matthews, 1999) where existing entities are discussed and modeled like classes as well as their relationship (associations), operators and attributes. To model these classes using itSIMPLE, the user must double-click on a class diagram on the tree view to open the diagram and to start dropping some classes and associations on it. Attributes, operators and associations properties are defined at the Element Property panel. The resulting Class Diagram is showed in Figure 6.

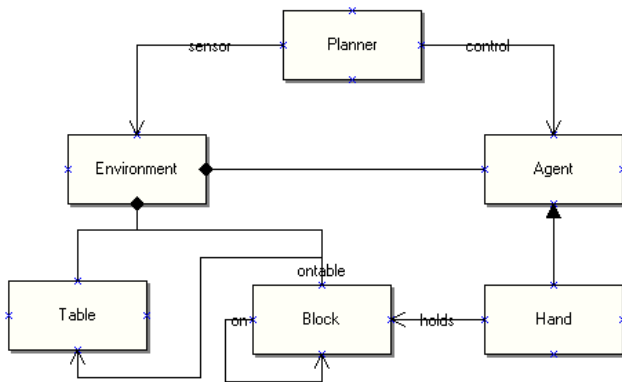


Figure 6 – Blocks domain modeled in itSIMPLE

A State Chart Diagram is necessary to model no static entities (in this case Block and Hand are no static classes). Again, a double-clicking on the tree view node diagram that represents state chart diagram allows the user to start dropping states and actions in order to model a preliminary dynamic behavior. The itSIMPLE just permits to specify in the State Chart Diagram those actions that were previously defined as operator in the Class Diagram. Since the Blocks World domain example has two dynamic classes, there are also two State Chart Diagrams in the model. Figure 7 shows the Hand State Chart Diagram.

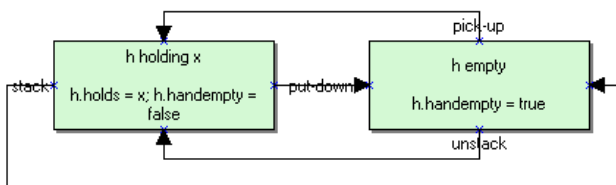


Figure 7 – Hand State Chart Diagram modeled in itSIMPLE

When Class Diagram and all State Chart Diagram are ready, the next step is to model the problem as an instance of the domain. In order to model a problem, the initial and the goal state must be specified in Object Diagrams. These diagrams are built by dropping and naming objects. Each object has its class which describes its features. When two objects are associated, itSIMPLE checks in the Class Diagram all possible associations between them; chooses the first possible association to be the current association and lists all possible associations in order to allow the user to change the current association established. The following figure shows a snapshot (initial) from Blocks World domain where there are three blocks, one table and one hand. Here, C is on B which is on A. Block A is on table and hand Hand1 is empty.

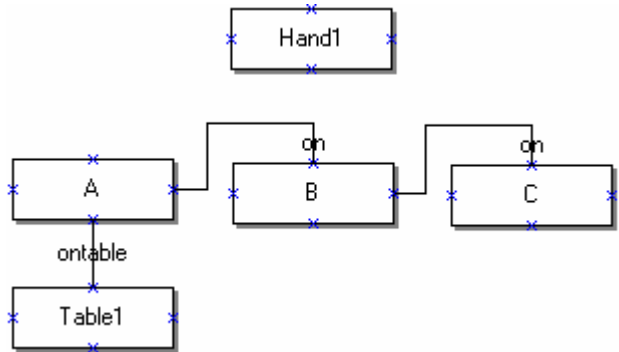


Figure 8 – Snapshot-init from a sample problem of Blocks World domain

Since the model is created, it can be saved as a XML file. UML can be easily translated to XML in many ways. One example, which is used by itSIMPLE, is showed as following. Each diagram is a tag in XML representation. The Class Diagram can be described:

```
<diagram name="BWDiagram" type="ClassDiagram" id="1" >
```

where the *type* attribute indicates the which UML diagram it is and *id* is used by itSIMPLE to search information by a fast fashion way.

Each diagram tag has classes and associations as sub-tags. Each class can have attributes, operators and generalizations as sub-tags. An example:

```
<class name="Table" id="1.7"/>
<class name="Block" id="1.8">
  <attribute name="clear" type="Boolean" id="1.8.1"/>
</class>
<class name="Hand" id="1.9">
  <attribute name="handempty" type="Boolean" id="1.9.1"/>
  <generalization element-type="class" element-id="1.2"
    id="1.9.2"/>
  <operator name="stack" id="1.9.5">
    <parameter name="h" element-type="class"
      element-id="1.9" id="1.9.5.1"/>
```

```

<parameter name="x" element-type="class"
  element-id="1.8" id="1.9.5.2"/>
<parameter name="y" element-type="class"
  element-id="1.8" id="1.9.5.3"/>
<parameter name="t" element-type="class"
  element-id="1.7" id="1.9.5.4"/>
</operator>
<operator name="unstack" id="1.9.6">
  <parameter name="h" element-type="class"
    element-id="1.9" id="1.9.6.1"/>
  <parameter name="x" element-type="class"
    element-id="1.8" id="1.9.6.2"/>
  <parameter name="y" element-type="class"
    element-id="1.8" id="1.9.6.3"/>
  <parameter name="t" element-type="class"
    element-id="1.7" id="1.9.6.4"/>
</operator>
</class>

```

Each operator has its own parameters that are linked to the diagram objects by element-type and element-id tag attributes.

Association nodes have usually two sub-tags that represent the association ends. Each association end holds multiplicity, role name, navigation and type. The navigation tag attribute is a boolean value that indicates the direction of the association. The false value indicates the origin of the association and true value indicates its target.

State Chart Diagram has some sub-tags such as actions and states. Actions are the operators defined in class diagram so they have a reference to those operators as well as they have references to states that they are linked to. The following XML code shows two states and the action “stack”.

```

<state name="h holding x" condition="h.holds = x; h.handempty
  =false" id="5.1"/>
<state name="h empty" condition="h.handempty = true"
  id="5.2"/>
<action element-type="operator" element-id="1.9.5"
  precondition="" postcondition="" id="5.6">
  <actionEnd element-type="state" element-id="5.1"
    navigation="false" id="5.6.1"/>
  <actionEnd element-type="state" element-id="5.2"
    navigation="true" id="5.6.2"/>
</action>

```

Object Diagram has children nodes such as object and object associations. The object tags are related to classes as they are mainly instance of classes. Object association has also reference to those associations defined in class diagram as it is an instance of them.

itSIMPLE uses mainly *strips* and *typing* requirements in PDDL, it can also support *fluents*. Types are easily extracted from classes following their generalizations. Functions are extracted from those class attributes that are declared as Numeric. For instance, a class *Table* can have an attribute called *load* defined as Numeric and it would appear in PDDL document in *:functions* section as (*load tab? - Table*).

Every boolean attribute and all associations from Class Diagram are represented as predicates in PDDL.

The name of the association becomes the name of the predicate. Association end with navigation equal to false becomes the first parameter of the predicate. The true value for navigation indicates the second parameter.

Operators declared for each class becomes actions in PDDL. From the Class Diagram it is possible to find the action names and their parameters. The State Chart Diagram gives us the whole features of pre and post-condition of an action. From the states, in each State Chart Diagram, we can extract part of the precondition and the delete list of the action, as well as some post-condition and add list predicates. It is made by a simple way: the origin state of an action provides precondition and delete lists predicates; the goal state of an action provides the post-condition and the insert list. In addition, the precondition and post-condition are complemented by the tag attributes of the action specification.

In one hand, both Class Diagram and State Chart Diagram provide necessary information to create a complete STRIPS-like domain specification in PDDL language.

On the other hand, the problem in PDDL language can be extracted from Object Diagrams. Two Object Diagrams are necessary to specify initial and goal states. They are the Snapshot-init and Snapshot-goal. Since they are specific tags in the XML representation, itSIMPLE can extract all the objects and declare them in *:objects* PDDL section.

In addition, all the instanced attributes and associations from the Snapshot-init are declared in *:init* section and all from Snapshot-goal are declared in *:goal* section. With these two diagrams itSIMPLE can model the problem as an instance of a domain.

Dynamic and static behaviors of Blocks World domain can be analyzed and validated using Petri Nets. Some works, such as Tate’s Common Editor Process (Tate et al, 1998) and Wilkins’s ACT editor (Myers and Wilkins, 1997) can provides important insights as this part of the project is still under development. In the problem example presented in Figure 8, a Petri Net should be built in order to begin the process of analyses. Here, it is depict only a final representation of the net schema since the Petri Net editor is not implemented yet in itSIMPLE. Figure 9 shows the problem and Figure 10 shows its Petri Net schema where it can be figured some aspects of the problem domain such as the relationship between the states. The closed world representation of blocks problem is such that it is possible to go from a special state (an essential node in the graph or Petri Net) to one of the three connected component moving either the block A, B or C. Thus, to plan to go from one state in one of the connected component to other connected component, it is necessary to have the essential node (all blocks on the table) as an intermediary goal. This can also avoid the Sussman anomaly (Sussman, 1990).

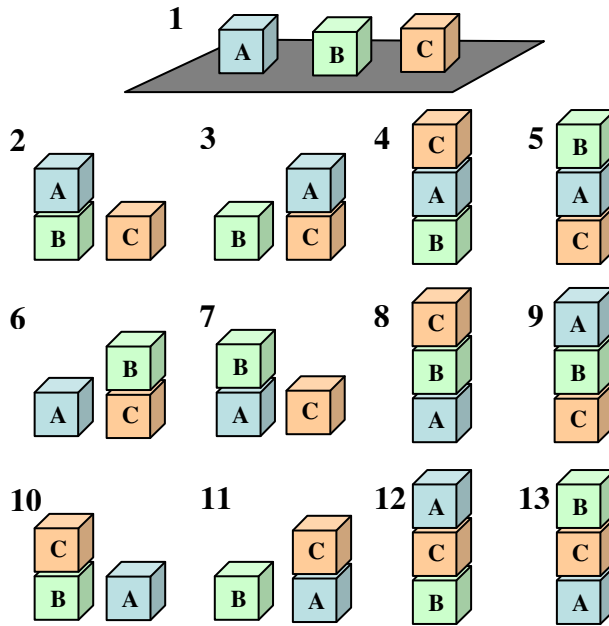


Figure 9 – Classical Blocks World problem – three blocks on a table

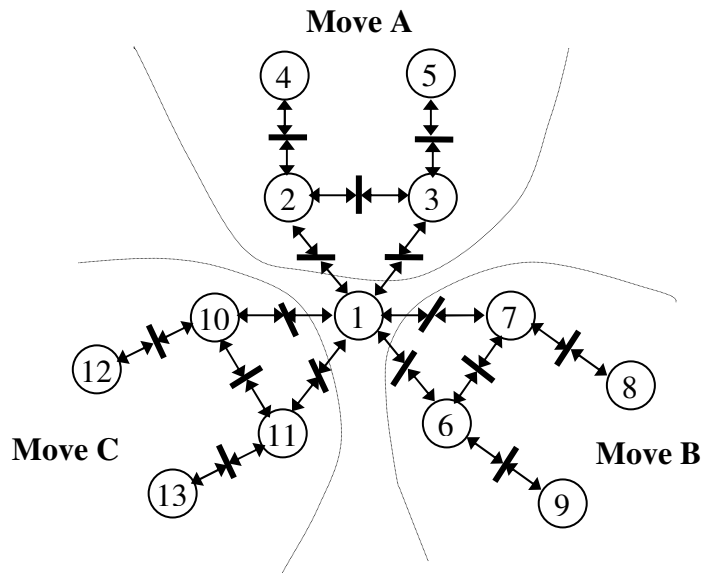


Figure 10 – Petri Net schema for the classical Blocks World problem

## Conclusion and Future Works

The software itSIMPLE proposed to ICKEP (International Competition on Knowledge Engineering for Planning) is

in its preliminary version, and just provides a simple environment based in UML models to express planning domains. However pure UML does not fit all structures in PDDL language, for instance. The fluent definition in PDDL language can be translated only if guided by new XML rules inserted in the translator component. The same occurs with Petri Nets and there a specific library called

PNML that adds in XML specific Petri Nets expression. In the future we will add to the translator a specific PDML (Planning Domain Modeling Language) to cover all this aspects. In the current version of itSIMPLE some rules as the ones for fluents were already included ad hoc.

The support for Petri Nets models, extracted from XML files, are under development and it is supported by some works in the literature (Silva and Santos 2004).

The next natural step is a validation algorithm that provides static and dynamic analysis of any planning domain. And a more distant target which is the classification of domain features, extracted from the domain analyses, in order to decide which kind of planning technique or heuristic suits better the proposed domain.

### Acknowledgements

The itSIMPLE project has been developed in the DLab (Design Lab) and IAAA Lab. The author would like to acknowledge the help of all coworker from both Labs for the helpful suggestions which have made this paper better.

### References

Bray, T.; Paoli, J.; McQueen, C.M.; Maler, E.; Yergeau, F. 2004. *Extensible Markup Language (XML) 1.0 – Third Edition*. Available in: <http://www.w3.org/TR/REC-xml/>.

D'Souza, F.D. and Wills, A.C. 1999. *Object, Components, and Frameworks with UML – The Catalysis Approach*. Addison-Wesley. United States of America and Canada.

Fox, M. and Long, D. 2003. *PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. Journal of Artificial Intelligence Research 20:61-124.

Murata, T. 1989. *Petri Nets: Properties, Analysis and Applications*. In Proceedings of IEEE, 77(4):541-580.

Scheetz, M., Mayrhauser, A., Dahlman, E. and Howe, A.E. 1999. *Generating Goal-oriented Test Cases*, In Proceedings COMPSAC '99, 110-115.

Simpson, R.M; T. L. McCluskey, W. Zhao, R. S. Aylett and C. Doniat 2001. *An Integrated Graphical Tool to support Knowledge Engineering in AI Planning*. Proceedings, 2001 European Conference on Planning, Toledo, Spain.

Silva, J R. and Santos, E. A. 2004. *Applying Petri nets to requirements validation*. In: IFAC Symposium on Information Control Problems in Manufacturing. Salvador, 2004. INCOM'04 : Salvador : IFAC, p. 1.

Sussman, G.J., 1990, The Virtuous Nature of Bugs, on Readings on Planning, J. Allen, J. Handler, A. Tate (eds.), Morgan Kauffman.

Vaquero, T. S., Tonidandel, F., Silva, J.R. 2005. *itSIMPLE: Integrated Tools Software Interface for Modeling Planning Environments*. Can be downloaded from <http://www.pmr.poli.usp.br/d-lab/site> or <http://dlab.poli.usp.br>.