itSIMPLE$_{2.0}$: An Integrated Tool for Designing Planning Domains

Tiago Stegun Vaquero[1] and **Victor Romero**[1] and **Flavio Tonidandel**[2] and **José Reinaldo Silva**[1]
[1]Escola Politécnica - Universidade de São Paulo
Design Lab, Mechatronic and Mechanical Systems Department - São Paulo, Brazil
[2]Centro Universitário da FEI
IAAA - Artificial Intelligence Applied in Automation Lab - São Bernardo do Campo, Brazil
{tiago.vaquero, victor.romero}@poli.usp.br, flaviot@fei.edu.br, reinaldo@usp.br

Abstract

A great effort has been made today in the area of Artificial Intelligence for defining reliable automated planning systems that can be applied in real life applications. That leads to the need of a systematic design process, in which the initial phases are not neglected and where Knowledge and Requirement Engineering tools have a fundamental role for supporting designers. Following this principle, this paper presents the evolution of the tool itSIMPLE which implements a KE integrated environment where designers can perform knowledge acquisition, domain modeling, domain model analysis, model testing, maintenance and plan analysis processes by using different well-known languages such as UML, Petri Nets, PDDL and XML, each one of them with its best contribution. The tool supports users in an organized object-oriented domain design process with a friendly and easy-to-use interface.

Introduction

The rising demand for reliable planning systems have become a great motivation to apply all developments already achieved in real life applications. This scenario leads to the need of a systematic design process, in which the initial phases are not neglected and where Knowledge and Requirement Engineering tools and methodologies have a fundamental role for supporting designers. These concepts have been addressed in the planning community in several initiatives, such as the first International Competition on Knowledge Engineering for Planning and Scheduling - ICKEPS 2005. This competition has brought extremely important knowledge and design engineering issues and showed powerful tools, such as itSIMPLE (Vaquero, Tonidandel, & Silva 2005), ModPlan (Edelkamp & Mehler 2005) and GIPO (Simpson 2005), all of them assisting designers to better understand, specify, visualize, verify and validate their planning domain models.

The itSIMPLE tool was designed to give support to users during the construction of a planning domain application mainly in the initial stages of the design life cycle. These initial stages encompass processes such as domain specification, modeling, analysis, model testing and maintenance, all of them crucial for the success of the application. The evolution of the itSIMPLE, called itSIMPLE$_{2.0}$, presents a enhanced integrated environment with well-known representation languages such as UML (OMG 2001), XML (Bray *et al.* 2004), Petri Nets (Murata 1989) and PDDL (Fox & Long 2003), each one of them with its best contribution to the whole design process, leading designers from the informality of real world requirements to formal domain models. Starting with requirements elicitation, specification and modeling, itSIMPLE proposes a special use of UML - Unified Modeling Language - in a planning approach (named UML.P) which we believe can contribute to the knowledge acquisition process (from different viewpoints) as well as to the domain model visualization and verification.

itSIMPLE$_{2.0}$ focuses also on the use of Petri Nets for dynamic domain analysis since it is a formalism with great potential for model checking and simulation. The tool also integrates PDDL representation to test models with general planners. In fact, by using itSIMPLE$_{2.0}$ it is possible to export a domain model to PDDL representation containing features from version PDDL3 (Gerevini & Long 2006). XML (Extended Markup Language) is suitably used as an intermediate language that can translate the model from UML to other representations such as PDDL or Petri Nets. Integrated in the initial stages, the environment also gives to end-users an interface for plan analysis and management where designers can observe the behavior of the model during the execution of plans (given by users or by planners). This is done by using variable observation in XY and Gantt charts.

itSIMPLE$_{2.0}$ is an open source project implemented in Java that provides a user-friendly GUI to model and analyze many planning domains at the same time. This fact usually contributes to domain model reusability and maintenance.

This paper is organized as follows: First, we describe the itSIMPLE$_{2.0}$ Environment and then we give a brief explanation about the integrated languages available. Next, we present each design process stage encompassed by the new itSIMPLE$_{2.0}$ environment such as requirements, modeling, model analysis, model testing and plan analysis. The paper ends with a conclusions for this work.

The itSIMPLE$_{2.0}$ Environment

The itSIMPLE$_{2.0}$ environment aims to help designers to overcome the problems encountered during life cycle of planning application projects, mainly at the requirements specification, modeling and analysis phases. The itSIMPLE$_{2.0}$ is designed to permit users to have a disciplined design process to create knowledge intensive models for several planning domains. The suggested process for designing planning domains follows a cyclic sequence of phases inherited from Software Engineering applications combined with modeling experiences acquired in the design of planning domain. Such process is shown in Figure 1.

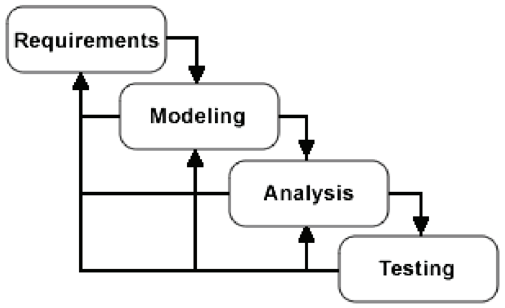


Figure 1: Planning domain design processes in itSIMPLE$_{2.0}$

The environment is independent of any existing planner, i.e., the processes of modeling, verification and dynamic domain validation are independent of any particular planning techniques. However, these techniques would be attached to the process analysis as a name list in order to be chosen (automatically or not) for a planning application.

The environment was designed to incorporate a toolset (representation languages and theories) capable of dealing with requirements and knowledge engineering as shown in Figure 1. Among many available specification languages, itSIMPLE$_{2.0}$ started by using the semi-formal language UML (which is a well-known diagrammatic language commonly used in the Requirement Engineering) for the requirement process and modeling. It is one of the most used language that models a great variety of applications and we believe that most engineers, working in several application areas, are somehow familiar with this representation. This fact makes UML a good choice for a modeling language in the planning domains context, mainly because of its suitability to make a first model (tracking requirement specifications). Since dynamic aspects are fundamental in planning approach, formal languages and theories that provide good representation of state space evolution may become an alternative to a sound planning domain specification. Following this principle, the environment provides the use of Petri Nets (Murata 1989), which is a formal representation, which can help designers to perform dynamic domain validations deploying visual and animated information to the entire dynamic system. By using Petri Nets it is possible to utilize all available techniques based on graphs in order to formally analyze the dynamic features of a planning domain (such as deadlocks, invariants, parallelism, concurrency and others). Also, since the AI Planning community has accepted the PDDL as the standard specification language for planner inputs, itSIMPLE integrates the capabilities of dealing with such language mainly in the model testing stage.

In order to hold all information available in several representation languages (UML, Petri Nets and PDDL) itSIMPLE uses the well-known language XML (Bray *et al.* 2004) which is commonly used in data transactions systems, web applications and data sharing systems. The important point on using XML is that some proposed integrated languages have direct representation in XML such as PNML - which stands for Petri Nets Markup Language (Billington *et al.* 2003) - for Petri Net representation and XPDDL - eXtensible Planning Domain Definition language (Gough 2002) - for PDDL. In itSIMPLE$_{2.0}$ all internal verifications and translations are performed in the structure and data available in the XML model which came from UML diagrams. In fact, in order to create a Petri Net representation, a model is first presented in PNML form and then showed to the user as a Petri Net graph for simulation and analysis. In the case of PDDL, all necessary data is extracted from XML specification in order to first translate the model to a XPDDL representation and then to PDDL. In fact, we believe XML can really help designers to share and maintain their domains models and knowledge. Considering all these issues, the integrated framework architecture of itSIMPLE$_{2.0}$ is shown in Figure 2.

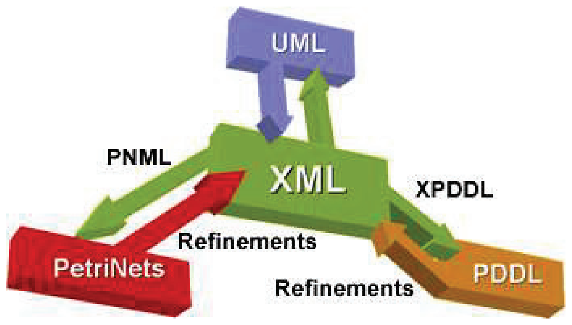


Figure 2: The architecture of the integrated languages

By using itSIMPLE$_{2.0}$ translators, designers are able to change from one language to another any time they want. Users can also deal with many projects and domains at the same time, which allows the reusability of models. In the following sections we present each main phase of the domain design process illustrated in Figure 1 using the integrated environment provided by itSIMPLE$_{2.0}$.

Requirements Elicitation

As Kotonya and Sommerville (Kotonya & Somerville 1996) states, ”requirements reflect the needs of customers and users of a system. They should include a justification for this system, what the system is intended to accomplish, and what design constraints are to be observed”. Truly, when dealing with real life planning application the requirements elicitation is one of the most important stage in the whole design process. During elicitation, knowledge engineers need to gather all the viewpoints from domain experts, users, stakeholders, sponsors and planning experts to investigate every aspect of the same problem.

Indeed, all the acquired requirements need to be documented and discussed exhaustively for saving time and resources in further stages. One of the well-known approaches for dealing and documenting requirements is the use case driven approach. By using use case diagrams from UML, it is possible to express requirements in a high level of abstraction. Following this approach, itSIMPLE$_{2.0}$ allows designers to build use case diagrams trying to specify the planning domain in a structured way. In these diagrams, each use case holds information such as descriptions, pre and post-conditions, flow events, invariants, constraints, issues and others relevant information that represents the domain requirements. Doing so, itSIMPLE$_{2.0}$ automatically generates an structured documentation which turns to be the principal reference for the early design phase. Figure 3 shows an example of the use case diagram for the classical planning domain Logistic.

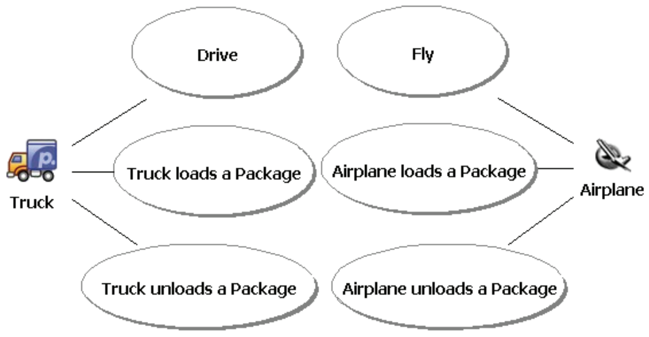


Figure 3: Use case driven approach for domain specification

Domain Modeling with UML.P

As highlighted before, in the itSIMPLE$_{2.0}$ environment, designers model their planning domains by using UML diagrams. The UML was first defined by OMG Unified Modeling Language Specification between 1996 and 1997 (D'Souza & Wills 1999), and nowadays is one of the most used languages to model a great variety of applications. Besides, UML has flexibility to attend many kinds of models in an object-oriented fashion since it is widely accepted as the standard for object-oriented analysis and design. This semi-formal modeling language is largely used for the modeling and visualization of system models. itSIMPLE$_{2.0}$ allows designers to use, in the planning domain context, all the collection of best engineering practices that are embedded in UML (OMG 2001).

Since UML is a general purpose modeling language it turns to be necessary to propose an extended representation that could deal with specification features that are intrinsically related to planning domains. It was called UML.P (UML in a Planning Approach) and was firstly introduced in (Vaquero, Tonidandel, & Silva 2005). For instance, some of the UML diagrams can be directly applied for planning domains such as class diagram, state chart diagram (also known as state machine diagram) and object diagram. In itSIMPLE$_{2.0}$ the designer can model a planning domain by using gradually these diagrams in order to model both domain and problem features as we depict in the following sections. Figure 4 shows a screenshot of itSIMPLE$_{2.0}$ interface for modeling activities.

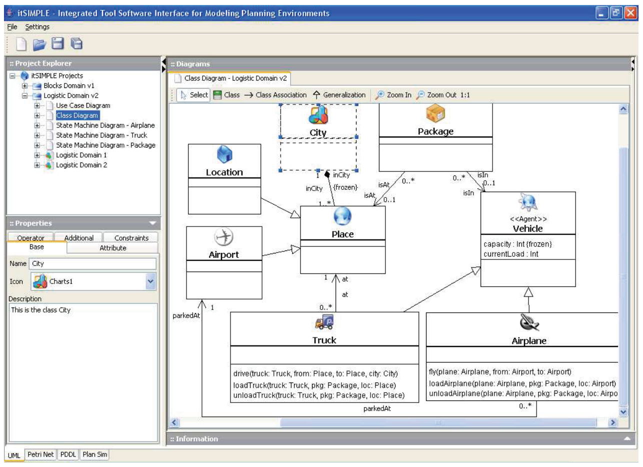

itSIMPLE - Integrated Tool Software Interface for Modeling Planning Environments
File Settings
:: Project Explorer
itSIMPLE Projects
Blocks Domain v1
Logistic Domain v2
Use Case Diagram
Class Diagram
State Machine Diagram - Airplane
State Machine Diagram - Truck
State Machine Diagram - Package
Logistic Domain 1
Logistic Domain 2
:: Properties
Operator Additional Constraints
Base Attribute
Name City
Icon Charts1
Description
This is the class City
:: Diagrams
Class Diagram - Logistic Domain v2
Select Class Class Association Generalization Zoom In Zoom Out 1:1
City
Package
Location
Place
Airport
<<Agent>>
Vehicle
capacity : Int {frozen}
currentLoad : Int
Truck
Airplane
inCity
{frozen}
isAt
isIn
at
parkedAt
1
1..*
0..*
0..1
drive(truck: Truck, from: Place, to: Place, city: City)
loadTruck(truck: Truck, pkg: Package, loc: Place)
unloadTruck(truck: Truck, pkg: Package, loc: Place)
fly(plane: Airplane, from: Airport, to: Airport)
loadAirplane(plane: Airplane, pkg: Package, loc: Airport)
unloadAirplane(plane: Airplane, pkg: Package, loc: Airpo
:: Information
UML Petri Net PDDL Plan Sim

Figure 4: itSIMPLE$_{2.0}$ interface for modeling with UML.P

Domain Features Modeling

By domain features modeling we mean the process of modeling not only the static structure of a planning domain - with the definition of classes, attributes, associations and structural constraints - but also the dynamic features such as states, actions an their pre and post conditions. In order to model these features in UML.P, two main diagrams are needed: the class diagram (for static features) and the state chart diagram (for dynamic characteristics).

The class diagram is the commonly used in object-oriented modeling process. itSIMPLE$_{2.0}$ provides a clean and very intuitive interface for modeling the main static structure of a planning domain, either a classical domain or a real life one, as depicted in (Vaquero *et al.* 2006). Beyond all the classes, attributes and association definitions, the designer can also specify which classes are capable of performing actions by declaring operators (each operator with its set of parameters). Assigning classes with their operators means that users are deciding which classes perform each action. Classes capable of performing actions are what we call the agent, while others are considered only resources in the model. Figure 5 illustrates an example of a class diagram for the Logistic domain.

Another diagram used for modeling the domain features is the state chart diagram. In UML.P the state chart diagram is responsible for representing dynamic features of the domain model. Such dynamic representation is actually the bottleneck in the planning domain modeling process.

In UML.P the designer build a state chart diagram for each class that has a dynamic behavior in the model. By defining one diagram for each dynamic class, users can view their model as a set of regions in the state space.

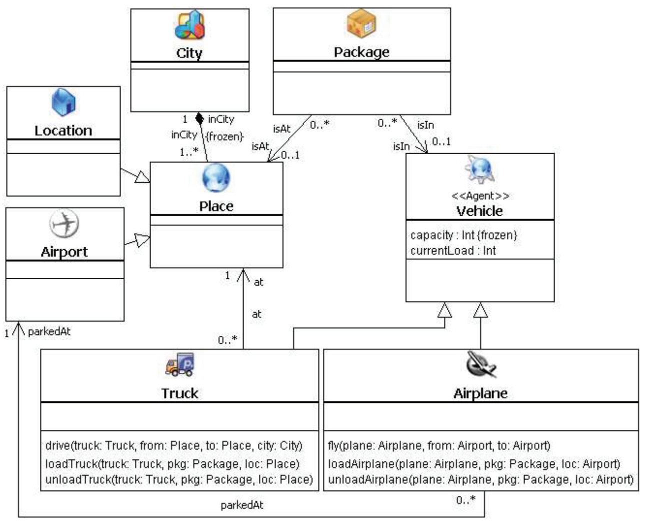

City
Package
Location
Place
Airport
<<Agent>>
Vehicle
capacity : Int {frozen}
currentLoad : Int
Truck
Airplane
drive(truck: Truck, from: Place, to: Place, city: City)
loadTruck(truck: Truck, pkg: Package, loc: Place)
unloadTruck(truck: Truck, pkg: Package, loc: Place)
fly(plane: Airplane, from: Airport, to: Airport)
loadAirplane(plane: Airplane, pkg: Package, loc: Airport)
unloadAirplane(plane: Airplane, pkg: Package, loc: Airport)
inCity
{frozen}
isAt
isIn
at
parkedAt
1
1..*
0..*
0..1

Figure 5: Modeling static features with class diagram

User specifies in the state chart diagram all the pre and post conditions of actions using the language OCL (Object Constraint Language) (OMG 2003) following the same approach presented in (D'Souza & Wills 1999). OCL is a predefined formal language of the UML to represent constraints, invariants and pre and post conditions of actions. In itSIMPLE$_{2.0}$, states are defined by using OCL expressions representing the possible values of attributes. This expressions are also used in transition arcs which represents an action in the chart. A complete action representation is the result of the union of all pre and postcondition expressions declared in all state chart diagram where such action appears. itSIMPLE$_{2.0}$ helps the user with an OCL editor to avoid mistakes while writing such expression. Figure 7 and Figure 6 show two state chart diagrams representing the dynamic features of two classes of the Logistic domain.

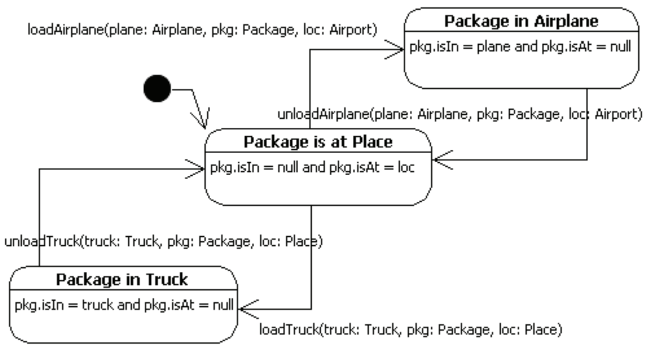

loadAirplane(plane: Airplane, pkg: Package, loc: Airport)
Package in Airplane
pkg.isIn = plane and pkg.isAt = null
unloadAirplane(plane: Airplane, pkg: Package, loc: Airport)
Package is at Place
pkg.isIn = null and pkg.isAt = loc
unloadTruck(truck: Truck, pkg: Package, loc: Place)
Package in Truck
pkg.isIn = truck and pkg.isAt = null
loadTruck(truck: Truck, pkg: Package, loc: Place)

Figure 6: State chart diagram for the Package class

We considered a great advantage to construct state chart diagrams for representing the dynamic features. However, since the designer can use more than one diagram to represent the dynamic aspect, it becomes difficult to guarantee that the domain model preserves the coherency among all

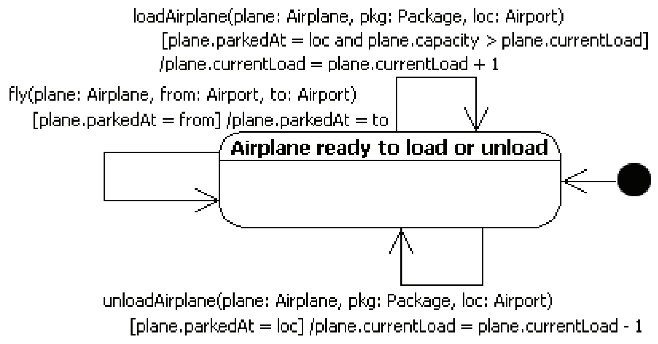

loadAirplane(plane: Airplane, pkg: Package, loc: Airport)
[plane.parkedAt = loc and plane.capacity > plane.currentLoad]
/plane.currentLoad = plane.currentLoad + 1
fly(plane: Airplane, from: Airport, to: Airport)
[plane.parkedAt = from] /plane.parkedAt = to
Airplane ready to load or unload
unloadAirplane(plane: Airplane, pkg: Package, loc: Airport)
[plane.parkedAt = loc] /plane.currentLoad = plane.currentLoad - 1

Figure 7: State chart diagram for the Airplane class

diagrams.

Unfortunately, the UML does not have a diagram that can represent the global states of the planning domain that would give the designer an entire view of the domain dynamic aspects. For this reason, itSIMPLE$_{2.0}$ provides a translation process from the state chart diagrams into Petri Nets (PN) in order to allow designer to verify and validate the coherence and consistency of his/her domain model. The main advantage on using PN is its formalism which provides simulation mechanisms and analysis techniques like invariants identification, deadlocks and others illustrated in (Murata 1989). Some works show the successful application of PN for validation and verification of dynamic aspects of UML models (Watanabe *et al.* 1997) (Cheung, Chow, & Cheung 1998) (Saldhana & Shatz 2000). The use of PN for analysis purpose is described further in the domain analysis section.

In order to complete the domain features modeling process using itSIMPLE$_{2.0}$, designer can represent the objects that compose the domain, i.e., the agents and the resources that will be used to model the planning problems. For that, users can utilize object diagrams as a repository of objects to be used in all problems.

Problem Features Modeling

A problem statement in a planning domain is usually characterized by a situation where only two states are known: the initial and goal state. The diagram used to describe these states is the object diagram or Snapshots. A good definition of Snapshot is given as a depiction of a set of objects and the values of some or all of their attributes at a particular point of time (D'Souza & Wills 1999).

In other words, a snapshot is a picture of a specific state representing an instantiation of the domain structure. Such instantiation represents features such as: which are the objects available in the problem (including agents and resources); how they are arranged; what the values of each object attribute are; and how the objects are related with each other. The itSIMPLE$_{2.0}$ tool assists designer to specify consistent snapshots by promoting automatic verification on all the constraints and features defined in the class diagram. For example, when two objects are associated, firstly itSIMPLE$_{2.0}$ verify all possible associations between them in the class diagram and then verify the multiplicity rules.

This available automatic verification in snapshot modeling can avoid inconsistent states. Figure 8 shows an example of a initial snapshot of a Logistic problem. This figure also illustrates the importance of using icons when modeling both domain and problem. Usually, icons and images provide an intuitive view of all problem components.

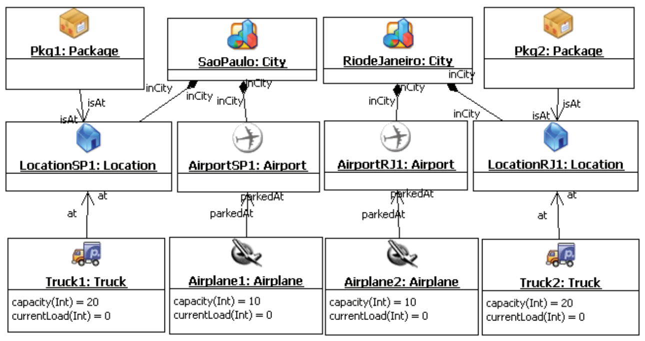


Figure 8: The initial state of a Logistic problem

Another important feature available in itSIMPLE$_{2.0}$ is the definition of intermediate states (snapshots) in a problem. These intermediate states can represent situations that must be avoided or some kind of state trajectory constraints during the planning process. This capability follows the concepts introduced in the definition of PDDL3 (Gerevini & Long 2006). Thus, the use of itSIMPLE$_{2.0}$ make it possible to model situations that must happen always, sometimes, at-most-once or never during the evaluation of a planning problem. This particular tool capability can be a powerful modeling characteristic for many real planning problems. All these constrained situations are considered snapshots in the tool environment.

Domain Model Analysis with Petri Nets

Domain analysis process is becoming one of the main studied topic in the KE for AI Planning, and, indeed, this process has a great impact on the quality of the domain model being built. As mentioned before, itSIMPLE$_{2.0}$ also provides mechanisms for helping designers to analyze and verify their model focusing on dynamic aspects. It is done by representing state chart diagrams into Petri Nets. As highlighted in this paper and in (Vaquero *et al.* 2006), each state chart diagram shows the dynamic characteristics of objects (of a specific class) being affected by actions that can appear in many state chart diagrams. Therefore, there are usually many state chart diagrams in a single domain model.

In this framework, each state chart diagram can be viewed as a module of the entire domain. Considering this modularity introduced by the state chart diagrams in UML, it is possible to represent these modules through the concept of *modular PNML* (Kindler & Weber 2001). This concept is used by itSIMPLE$_{2.0}$ to represent Petri Nets. It is important to highlight that OCL expressions are not considered in the PNML representation in itSIMPLE$_{2.0}$.

The dynamic domain analysis process using the state charts and Petri Nets is divided in two main analyses: *Modular Analysis* and *Interface Analysis*.

Modular Analysis

The *Modular Analysis* tries to verify each module one-by-one, taking into account the dependencies with others modules (represented by what we call Gates). The modular analysis through Petri Nets allows the designer to detect features like parallelism, concurrencies and undesirable deadlocks throughout simulation processes. Figure 9 and Figure 10 show an example of Petri Nets derived from the state charts illustrated in Figure 6 and Figure 7, respectively.

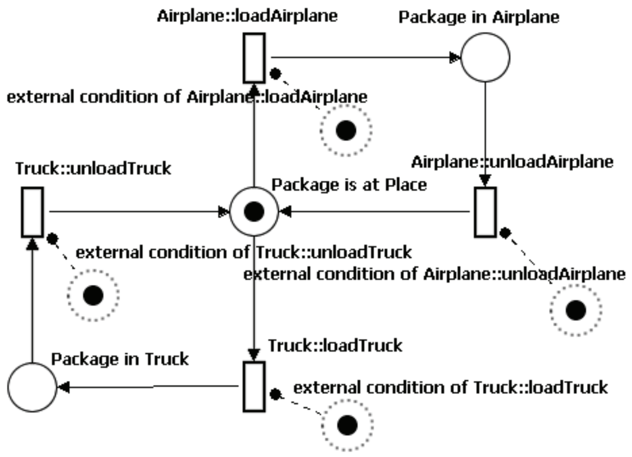

Airplane::loadAirplane
Package in Airplane
external condition of Airplane::loadAirplane
Truck::unloadTruck
Airplane::unloadAirplane
Package is at Place
external condition of Truck::unloadTruck
external condition of Airplane::unloadAirplane
Truck::loadTruck
Package in Truck
external condition of Truck::loadTruck

Figure 9: Modular Analysis of the class Package

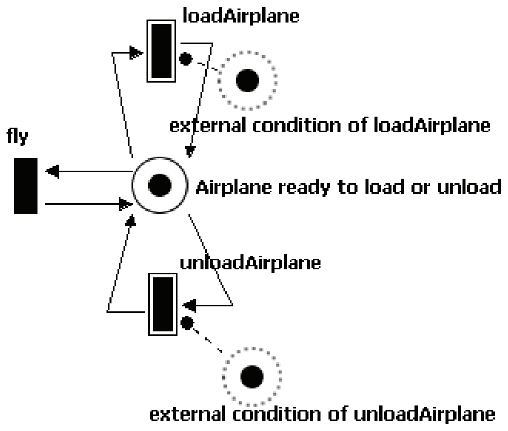

loadAirplane
external condition of loadAirplane
fly
Airplane ready to load or unload
unloadAirplane
external condition of unloadAirplane

Figure 10: Modular Analysis of the class Airplane

In order to analyze modules, designers must verify the flow of the marks in each one, checking the existence of undesirable deadlocks and also validating the concurrencies and parallelism by manipulating Gates (representing external dependencies from other modules). This manipulation (turning Gates on or off) allows the designer to realize what happens, for example, with a package when an airplane is not available. This modular visualization and manipulation are extremely important and very useful for the designer in a verification process, especially when dealing with complex and large domains.

Interface Analysis

The *Interface Analysis* has the purpose of investigating dynamic interactions among modules. During this analysis, designers can verify not just one module but many of them, all together, visualizing the dependencies among classes during actions domain executions. A particular and very important analysis is made when all modules are put together providing a dynamic view of the entire model. itSIMPLE$_{2.0}$ treats the union of all modules and presents the resulting Petri Net as shown in Figure 11 with modules from Figures 9 and 10.

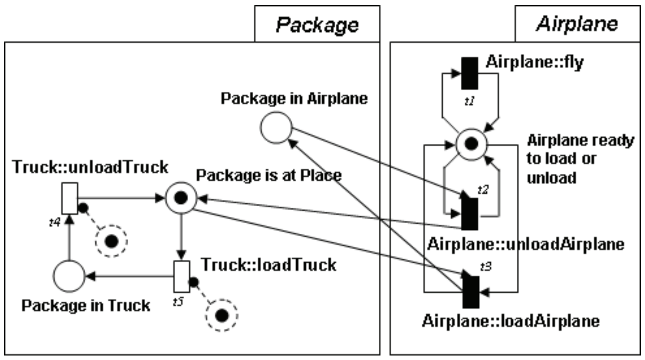


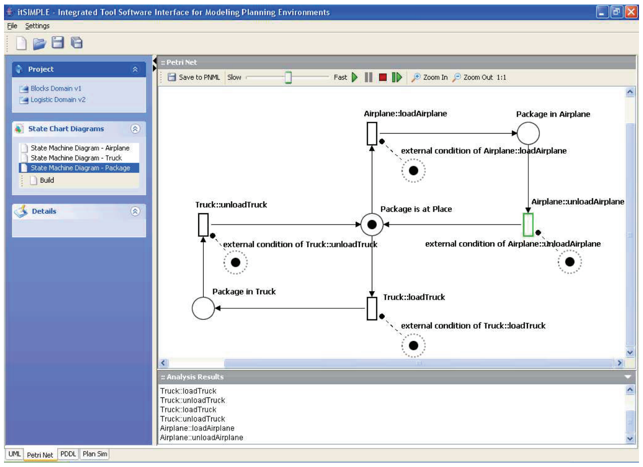
Figure 11: Interface Analysis of the classes Package and Airplane

itSIMPLE$_{2.0}$'s interface for dealing with dynamic domain analysis with Petri Nets is shown in Figure 12. In this interface, users can perform both modular and interface analyses by generating Petri Nets. These nets can be simulated and the sequence of fired transition can be observed in the provided console. Additional Petri Nets verification techniques are easily found in the literature (Murata 1989) (Christensen & Petrucci 2000) like analysis of invariants, reachability, and others. As mentioned before, the environment supports exportation of nets through PNML representation in order to use specific analysis techniques.

The successive refinements made in the model through Petri Nets are very important to the success of the final domain model and also to its matching to reality. Real applications could be very complex and a dynamic visualization, in a modular way like that presented in this paper, can avoid many mistakes in the modeling process of such applications. We believe that the dynamic domain model analyses can bring additional information that contributes to improve the planner search engine by pruning some undesirable states.

Model Testing and Plan Analysis

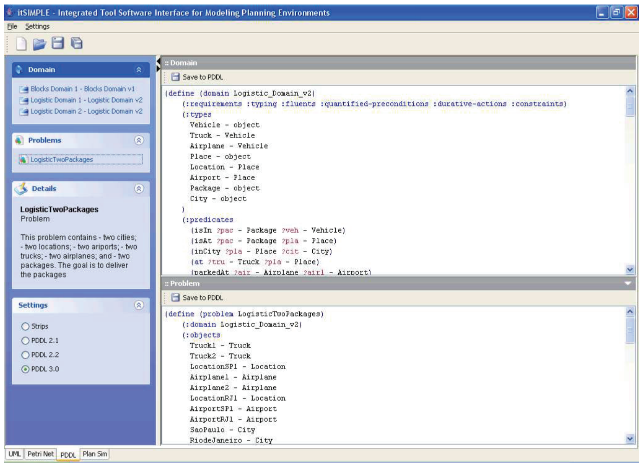
In order to perform model testing for verification and validation of the planning domain model, itSIMPLE$_{2.0}$ enables the user to represent UML models (holded in XML) in a PDDL format. As highlighted before, the tool transforms the


itSIMPLE - Integrated Tool Software Interface for Modeling Planning Environments
File
Settings
Project
Blocks Domain v1
Logistic Domain v2
State Chart Diagrams
State Machine Diagram - Airplane
State Machine Diagram - Truck
State Machine Diagram - Package
Build
Details
:: Petri Net
Save to PNML
Slow
Fast
Zoom In
Zoom Out
1:1
Airplane::loadAirplane
Package in Airplane
external condition of Airplane::loadAirplane
Truck::unloadTruck
Package is at Place
Airplane::unloadAirplane
external condition of Truck::unloadTruck
external condition of Airplane::unloadAirplane
Package in Truck
Truck::loadTruck
external condition of Truck::loadTruck
:: Analysis Results
UML
Petri Net
PDDL
Plan Sim

```
Truck::loadTruck
Truck::unloadTruck
Truck::loadTruck
Truck::unloadTruck
Airplane::loadAirplane
Airplane::unloadAirplane
```

Figure 12: The dynamic analysis with Petri Nets interface

model into a XPDDL representation and then into a PDDL. itSIMPLE$_{2.0}$ allows designers to deal with features from PDDL2.1 and PDDL3 (Gerevini & Long 2006) such as general constraints and state trajectory constraints. However, itSIMPLE$_{2.0}$ do not deal with time constraints. itSIMPLE$_{2.0}$ builds the domain and the problem specification in PDDL separately. In order to build the PDDL specification to the domain, the tool extract types, predicates (attributes and associations) and functions (attributes) from the class diagram and the actions, as well as pre and postconditions from the OCL expression available at all state chart diagrams (some domain constraints are extract from associations rules). To build a problem specification in PDDL, itSIMPLE$_{2.0}$ extracts objects and the instantiated situation from the snapshots, including the state trajectory constraints. It also provides a PDDL editor for additional modeling features that the designer wants to include. itSIMPLE$_{2.0}$'s interface for dealing with PDDL is shown in Figure 13.


itSIMPLE - Integrated Tool Software Interface for Modeling Planning Environments
File
Settings
Domain
Blocks Domain 1 - Blocks Domain v1
Logistic Domain 1 - Logistic Domain v2
Logistic Domain 2 - Logistic Domain v2
Problems
LogisticTwoPackages
Details
Settings
:: Domain
Save to PDDL
:: Problem
Save to PDDL
UML
Petri Net
PDDL
Plan Sim

LogisticTwoPackages
Problem

This problem contains - two cities; - two locations; - two ariports; - two trucks; - two airplanes; and - two packages. The goal is to deliver the packages

Strips
PDDL 2.1
PDDL 2.2
[✓ PDDL 3.0]

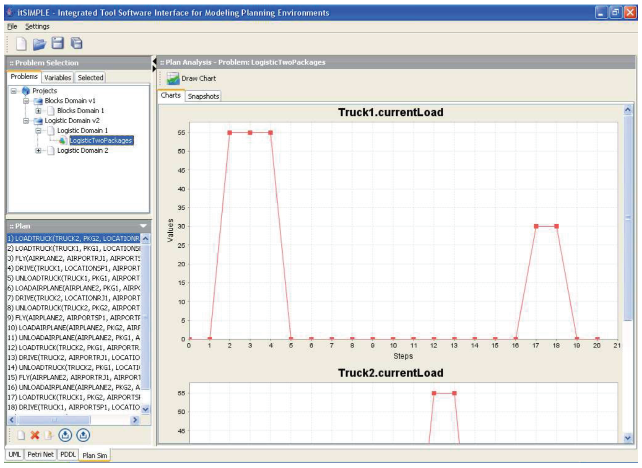
```
(define (domain Logistic_Domain_v2)
    (:requirements :typing :fluents :quantified-preconditions :durative-actions :constraints)
    (:types
      Vehicle - object
      Truck - Vehicle
      Airplane - Vehicle
      Place - object
      Location - Place
      Airport - Place
      Package - object
      City - object
    )
    (:predicates
      (isIn ?pac - Package ?veh - Vehicle)
      (isAt ?pac - Package ?pla - Place)
      (inCity ?pla - Place ?cit - City)
      (at ?tru - Truck ?pla - Place)
      (parkedAt ?air - Airplane ?airl - Airport)
```

```
(define (problem LogisticTwoPackages)
    (:domain Logistic_Domain_v2)
    (:objects
      Truck1 - Truck
      Truck2 - Truck
      LocationSP1 - Location
      Airplane1 - Airplane
      Airplane2 - Airplane
      LocationRJ1 - Location
      AirportSP1 - Airport
      AirportRJ1 - Airport
      SaoPaulo - City
      RiodeJaneiro - City
```

Figure 13: Interface for dealing with PDDL representation

One important phase in the design process that can be performed once the PDDL representation is available is the Plan Analysis. Designer's mistakes are common during the domain modeling, especially when modeling real systems. Besides, sometimes the final model doesn't truly represent what the designer intended to, or, it does not fit what is defined in the requirements. Thus, not always the generated plan will achieve the solution expected by the designer. However, even using general planners as the only interface for analysis, it is often difficult to tell, only by looking at a set of actions in text format, whether the plan really represents the solution for a real problem or whether the domain agents and resources are being well used, or either if a constraint is being violated. In this way, itSIMPLE$_{2.0}$ intends to make the verification of plans easier and more productive.

The tool provides two ways of performing plan analysis. The first one is made through the use of XY and Gantt charts in what we call the *Analysis and Tracking of Variables*. The second one is made by observing screenshots in the same way as seeing a movie which starts from the problem initial states and goes to the goal state, shot by shot. This second process we call *Movie Maker*. itSIMPLE$_{2.0}$'s interface for dealing with Plan Analysis is shown in Figure 14.



:: Plan

1) LOADTRUCK(TRUCK2, PKG2, LOCATIONR
2) LOADTRUCK(TRUCK1, PKG1, LOCATIONSI
3) FLY(AIRPLANE2, AIRPORTRJ1, AIRPORTS
4) DRIVE(TRUCK1, LOCATIONSP1, AIRPORT
5) UNLOADTRUCK(TRUCK1, PKG1, AIRPORT
6) LOADAIRPLANE(AIRPLANE2, PKG1, AIRP(
7) DRIVE(TRUCK2, LOCATIONRJ1, AIRPORT
8) UNLOADTRUCK(TRUCK2, PKG2, AIRPORT
9) FLY(AIRPLANE2, AIRPORTSP1, AIRPORTF
10) LOADAIRPLANE(AIRPLANE2, PKG2, AIRF
11) UNLOADAIRPLANE(AIRPLANE2, PKG1, A
12) LOADTRUCK(TRUCK2, PKG1, AIRPORTR.
13) DRIVE(TRUCK2, AIRPORTRJ1, LOCATIO
14) UNLOADTRUCK(TRUCK2, PKG1, LOCATI(
15) FLY(AIRPLANE2, AIRPORTRJ1, AIRPORT
16) UNLOADAIRPLANE(AIRPLANE2, PKG2, A
17) LOADTRUCK(TRUCK1, PKG2, AIRPORTSI
18) DRIVE(TRUCK1, AIRPORTSP1, LOCATIO

Figure 14: itSIMPLE's Plan Analysis interface

For the *Analysis and Tracking of Variables* the user can not only look at a Gantt Chart (also used in others KE tools such as ModPlan (Edelkamp & Mehler 2005)) for checking plan sequence but also track and visualize the values of variables in XY charts during the execution of the whole plan. In this way designers can verify the possible violation of domain constraints and requirements while validating the attributes values, the state flow and the association among objects. Figure 15 illustrates a Gantt chart for a Logistic domain model and Figure 16 illustrates the XY charts for observing variables such as the current load of an airplane and a truck during the plan execution.

The user can also define its own plan for Plan Analysis by giving it to itSIMPLE$_{2.0}$, action by action. Some works also addressed the relevance of this flexibility (Simpson 2005). In fact, the designers usually refine the model while performing such analysis by including additional constraints

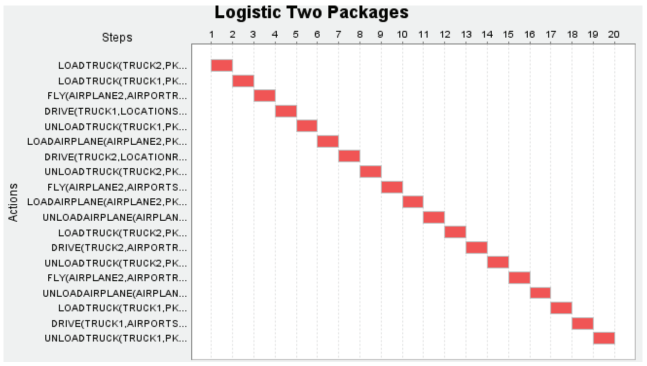


Figure 15: Simple example of Gantt Chart

into the model for avoiding, for example, undesirable states. In the *Movie Maker* analysis the user observes the screenshots as UML snapshot diagram going forward or backward when necessary.

Conclusion

The idea of a Knowledge Engineering environment for designing of real world planning problem is promising and appropriated to the current scenario, where new domain-independent planners are evolving and being tested to encompass situations more and more complex. The itSIMPLE$_{2.0}$ considers a design life cycle for real planning problems based on the use of well-known languages and tools utilized in real domain specification in an object-oriented approach. All remaining features could be composed and unified, once they could be expressed in a language that is compatible with XML.

Such flexibility of itSIMPLE$_{2.0}$ makes it an interesting environment for gathering requirements, domain modeling, model analysis, model testing and also as a testbed for new formal representations for planning.

itSIMPLE$_{2.0}$ is in constant improvement seeking for requirements and knowledge engineering aspects for bridging the gap between real planning problems and available planning techniques where it is not possible to neglect the initial phase of the design process. The presented environment still aims to fulfill as much as possible the knowledge engineering requirements in order to send valuable information extracted from the domain experts, planning experts and from the model itself and pass it to the planning systems. Some of future works are related to automatic action learning methods and capabilities, like (Yang, Wu, & Jiang 2007), and temporal representation.

The itSIMPLE tool is available in the projects link on the DesignLab web page http://designlab.grupoe.com.br.

Acknowledgments

The authors are very grateful to all Design Lab and IAAA members the enormous help in the itSIMPLE project, specially to the students Fernando Sette for all support and implemented features. We would like to thank FAPESP and Capes for the financial support.

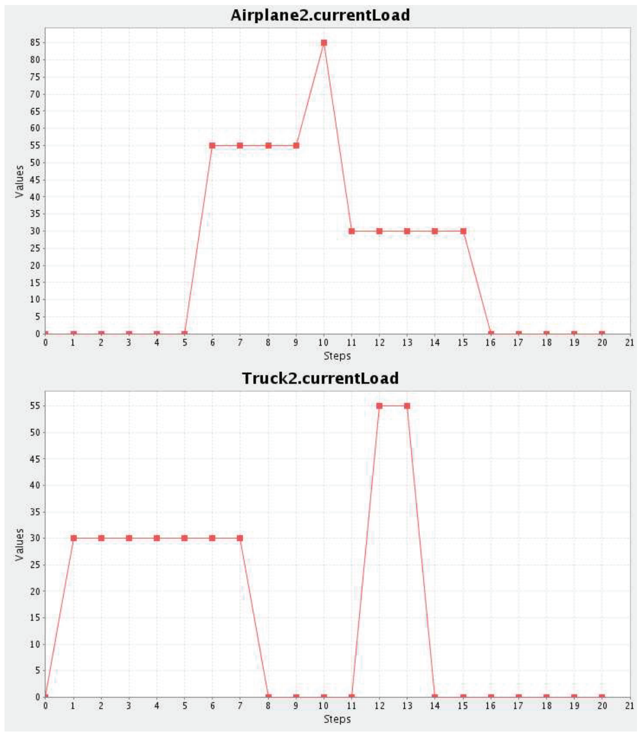


Figure 16: Example of analysis with XY charts for two numeric attributes

References

Billington, J.; Christensen, S.; van Hee, K.; Kindler, E.; Kummer, O.; Petrucci, L.; Post, R.; Stehno, C.; and Weber, M. 2003. The petri net markup language: concepts, technology, and tools. In *Proceedings of the 24th Int Conf on Application and Theory of Petri Nets, LNCS 2679, Springer*, 483–505.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2004. Extensible Markup Language (XML) 1.0 (Third Edition). Technical report.

Cheung, K. S.; Chow, K. O.; and Cheung, T. Y. 1998. Deriving scenarios of object interaction through petri net. In *TOOLS '98: Proceedings of the Technology of Object-Oriented Languages and Systems*, 104. Washington, DC, USA: IEEE Computer Society.

Christensen, S., and Petrucci, L. 2000. Modular analysis of Petri nets. *The Computer Journal* 43(3):224–242.

D'Souza, D. F., and Wills, A. C. 1999. *Objects, components, and frameworks with UML: the catalysis approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Edelkamp, S., and Mehler, T. 2005. Knowledge acquisition and knowledge engineering in the modplan workbench. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, Califormia, USA*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension of pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in pddl3. In Gerevini, A., and Long, D., eds., *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, 46–53.

Gough, J. 2002. Xpddl 0.1b: A xml version of pddl.

Kindler, E., and Weber, M. 2001. A universal module concept for petri nets. In *Proceedings of the 8th Workshops Algorithmen und Werkzeuge fr Petrinetze*, 7–12.

Kotonya, G., and Somerville, I. 1996. Requirements engineering with viewpoints.

Murata, T. 1989. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, 541–580.

OMG. 2001. *OMG Unified Modeling Language Specification, m Version 1.4*.

OMG. 2003. *OMG Unified Modeling Language Specification - Object Constraint Language m Version 2.0*.

Saldhana, J. A., and Shatz, S. M. 2000. Uml diagrams to object petri net models: An approach for modeling and analysis. In *Twelfth International Conference on Software Engineering and Knowledge Engineering, Chicago, IL, USA*.

Simpson, R. M. 2005. Gipo graphical interface for planning with objects. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, Califormia, USA*.

Vaquero, T. S.; Tonidandel, F.; Barros, L. N.; and Silva, J. R. 2006. On the use of uml.p for modeling a real application as a planning problem. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 434–437.

Vaquero, T. S.; Tonidandel, F.; and Silva, J. R. 2005. The itsimple tool for modelling planning domains. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, California, USA*.

Watanabe, H.; Tokuoka, H.; Wu, W.; and Saeki, M. 1997. A technique for analysing and testing object-oriented software using coloured petri nets. *IPSJ SIGNotes Software Engineering* 117.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artif. Intell.* 171(2-3):107–143.