EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization

avier Barreiro, Matthew Boyce, Jeremy Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Morris, Tristan Smith, Minh D

Abstract

*EUROPA*is a class library and tool set for building and analyzing planners (and/or schedulers) within a Constraint-based Temporal Planning paradigm. This paradigm has been successfully applied in a wide range of practical planning problems and has a legacy of success in NASA applications. *EUROPA* offers capabilities in 3 key areas of problem solving: (1) Representation; (2) Reasoning; and (3) Search. *EUROPA* is a means to integrate advanced planning, scheduling and constraint reasoning into an end-user application and is designed to be open and extendable to accommodate diverse and highly specialized problem solving techniques within a common design framework and around a common technology core. In this paper, we will demonstrate the core capabilities of this open-source planning & scheduling framework. While *EUROPA* is the complete planning and scheduling software suite, we will concentrate on the aspects that are relevant to the knowledge engineering purpose: modeling language capabilities, model debugging and execution tools, and plan visualization and analysis.

Introduction

EUROPA (Extensible Universal Remote Operations Planning Architecture) is a class library and tool set for building planners (and/or schedulers) within a Constraint-based Temporal Planning paradigm (**?**). Constraint-based Temporal Planning (and Scheduling) is a paradigm of planning based on an explicit notion of time and a deep commitment to a constraint-based formulation of planning problems. This paradigm has been successfully applied in a wide range of practical planning problems and has a legacy of success in NASA applications including :

- Observation scheduling for the Hubble Telescope (**?**)
- Autonomous control of DS-1.
- Ground-based activity planning for MER.
- Autonomous control of EO-1.

EUROPA is now at version 2.6 and is the successor of the original *EUROPA* which in turn was based upon HSTS (**?**). It has been made available under an open-source license. The source code and extensive documents on *EUROPA* are available at: *http://code.google.com/p/europa-pso/*. As a Planning & Scheduling Knowledge Engineering tool, *EUROPA* major strengths are: (1) flexibility in integrating with client applications; (2) proven track record; (3) open-source software license; (4) online document repository with detailed guidelines and a variety of examples in different domains.

EUROPA offers capabilities in 3 key areas of problem solving:

1. **Representation**: *EUROPA* allows a rich representation for actions, states, resources and constraints that allows concise declarative descriptions of problem domains and powerful expressions of plan structure. This representation is supported with a high-level object-oriented modeling language for describing problem domains and data structures for instantiating and manipulating problem instances.
2. **Reasoning**: Algorithms are provided which exploit the formal structure of problem representation to enforce domain rules and propagate consequences as updates are made to the problem state. These algorithms are based on logical inference and constraint-processing. Specialized techniques for reasoning about temporal quantities and relations included in *EUROPA* are particularly useful to deal with real-life problem domains.
3. **Search**: Problem solving in *EUROPA* requires search. Effective problem solving typically requires heuristics to make search tractable and to find good solutions. *EUROPA* provides a framework for integrating heuristics into a basic search algorithm and for developing new search algorithms.

EUROPA is not an end-user application. Rather, it is a means to integrate advanced planning, scheduling and constraint reasoning into an end-user application. *EUROPA* is not a specific planner or a scheduler. Rather it is a framework for developing specific planners and/or schedulers. It is designed to be open and extendable to accommodate diverse and highly specialized problem solving techniques within a common design framework and around a common technology core.

EUROPA is unconventional in providing a separate Plan Database that can integrate in a wide variety of applications. This reflects the common needs for representation and manipulation of plan data in different application contexts and different problem solving approaches. Possible approaches include:

- A batch planning application where an initial state is input and a final plan is output without any interaction with other actors.
- A mixed-initiative planning application where human users interact directly with a plan database but also employ an automated problem solver to work on parts of the planning problem in an interleaved fashion.
- An autonomous execution system where the plan database stores the plan data as it evolves in time, being updated from data in the environment, commitments from the executive, and the accompanying automated solver which plans ahead and fixes plans when they break.

While *EUROPA* is a large complex framework which provides many reasoning capabilities beyond the typical planning knowledge engineering tool, in this paper we will concentrate on aspects that are more relevant to IKEPS such as: modeling capabilities, model debugging and execution tools, and plan visualization and analysis. To emphasize the strength of *EUROPA* as a general purpose planning & scheduling framework, we will include examples from different class of problems such as resource scheduling, simple planning domains (BlocksWorld), complex planning domains (Rovers), and CSP benchmarks (N-Queens)[1]. To give a more complete picture of the whole system, we will also provide a brief background on *EUROPA* main architecture and how key components are integrated.

For the rest of this paper, we will first outline the modeling and reasoning capabilities. We then provide a short guide on how to use *EUROPA* in the most effective way and illustrate it with a list of simple examples. We then list the NASA and non-NASA projects that have used *EUROPA*. We finish the paper with a brief discussion of related work and discussion of our product roadmap for future releases of *EUROPA*.

Technical Background

In this section, we will start with an introduction to *EUROPA* main modeling language with concentration on its modeling capabilities. We then follow with the brief description on *EUROPA* architecture and its key components. This will set the background for subsequent sections on knowledge engineering tools that are provided as part of the *EUROPA* distribution to assist with both the front-end (modeling assistant) and back-end (plan execution, visualization, and analysis) JRB: is this a standard definition of front-end back-end in KE?, doesn't strike me as intuitive.

Modeling in NDDL

EUROPA's main input modeling language is New Domain Definition Language (NDDL) (pronounced 'noodle'), a domain description language for constraint-based planning and scheduling problems. NDDL can describe a number of concepts based on Variables and Constraints [2]. The NDDL representation includes state and activity descriptions, as is common in planners using traditional modeling languages like the Planning Domain Definition Language (PDDL) (**?**; **?**). However, unlike PDDL, NDDL uses a state variable-value formalism. *EUROPA* thus takes its heritage from planning formalisms like IxTeT (**?**) and SAS (**?**). *EUROPA* state variables are called timelines, and the values of timelines are sequences of states. States are temporally extended predicates, and consist of a proposition and a list of parameters, which by default includes the start, end and duration times. Timelines are totally ordered sequences of states; hence, a timeline can be in only one state at any instant. The final component of NDDL model is a set of compatibilities that govern the legal arrangements of states on, and across, timelines. These compatibilities are logical implications asserting that if a timeline is in a state, then other timelines must be in one of a set of compatible states. Compatibilities can incorporate explicit constraints on the parameters of the states. *EUROPA* provides a library of such constraints, and this library can be extended if new constraints are needed. JRB: Timelines are a specific instance of the more general concept of Objects in europa, however, for this audience this description is probably appropriate

There are several examples of NDDL for well known planning and CSP domains such as Blocksworld, 8-Queen, RCPSP available at the *EUROPA* website. JRB: do we want to add a short nddl snippet that show the concepts mentioned above?

The NDDL Transaction Language: NDDL includes procedural extensions, referred to as the *NDDL Transaction Language*, to operate on the plan database and thus initialize and/or modify a partial plan. A design goal of the NDDL transaction language is to provide syntax and semantics closely related to the use of NDDL elsewhere for class, predicate and rule declaration. However, the NDDL transaction language pertains exclusively to run-time data. It is referred to as a transaction language since a set of statements in this language form a procedurally executed sequence of atomic operations on the plan database, which stores an instance of a partial plan. Each statement of the language is thus directly translated into one or more operations available through the DbClient interface. The NDDL transaction language has many applications. The most common one is the construction of an initial partial plan as an input to a solver. A second important application is to log transactions on the plan database for later replay. This is useful for copying a database, and for reproducing a state found through planning in a direct manner without having to search. It is also a potentially very useful integration mechanism for pushing updates to the plan database from external systems.

[1]All examples are included in the open-source distribution of *EUROPA*

[2]A complete NDDL Reference guide with examples is available at: *http://code.google.com/p/europa-pso/wiki/NDDLReference* and the NDDL grammar guide is available at: *http://code.google.com/p/europa-pso/source/browse/PLASMA/trunk/src/PLASMA/NDDL/base/antlr/NDDL3.g*

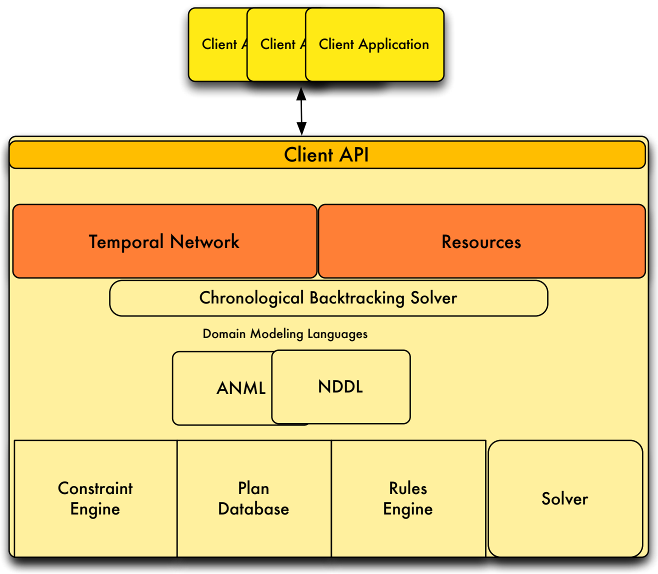


Figure 1: *EUROPA* Architecture

JRB: do we want to add a short nddl snippet that show the concepts mentioned above?

Supports for other Modeling Language: At the current moment, we are adding support for ANML (**?**), the new modeling language that import features from PDDL, IxTeT, AML, and NDDL. Given that there are published works on the possible translation between ANML and PDDL (**?**; **?**), we also plan to write the translator to support PDDL through ANML translation.

EUROPA's Main Components

Figure **??** shows the main components of *EUROPA* and the relationships/interactions between them. JRB: I replaced the figure with one that's more up to date, but we still need to include the UI and eclipse tools in it

JRB: I'll have to update this section

Utils module: provides common C++ utility classes for error checking, smart pointers etc. It also includes a very useful debugging utility. Many common programming practices in *EUROPA* development are built on assets in this module.

Constraint Engine: is the nexus for consistency management. It provides a general-purpose component-based architecture for handling dynamic constraint networks. It deals in variables and constraints. It includes an open propagation architecture making it straightforward to integrate specialized forms of local and global constraint propagation.

Plan Database: adds higher levels of abstractions for tokens and objects and the interactions between them. This is the code embodiment of the *EUROPA* planning paradigm. It supports all services for creation, deletion, modification and inspection of partial plans. It maintains the dynamic constraint network underlying a partialplan by delegation to the Constraint Engine and leverages that propagation infrastructure to maintain relationships between tokens and objects.

Solvers module: provides abstractions to support search in line with the *EUROPA* planning approach. It includes a component-based architecture for Flaw Identification, Resolution and heuristics as well as an algorithm for chronological backtracking search. As additional search algorithms are implemented they will be added to this module.

Rules Engine: provides the inference capabilities based on domain rules described in the model. It is almost exclusively used to execute NDDL rules but can be extended for custom rule formats.

Resources module: provides specialized algorithms and data structures to support metric resources (e.g. battery, power bus, disk drive). The Temporal Network module provides specialized algorithms and data structures to support efficient propagation of temporal constraints.

NDDL module: provides a parser and compiler for NDDL (pronounced noodle) which is a very high-level, object-oriented, declarative domain and problem description language. This module defines the mapping from the language to the code and consequently interfaces to a number of key modules in the system.

From an application developers view-point, the modules of interest are: *NDDL*, *Solvers* and JRB: UI and eclipse tools?. These modules address modeling, search and troubleshooting respectively. Other modules will be explored in the context of making customized extensions.

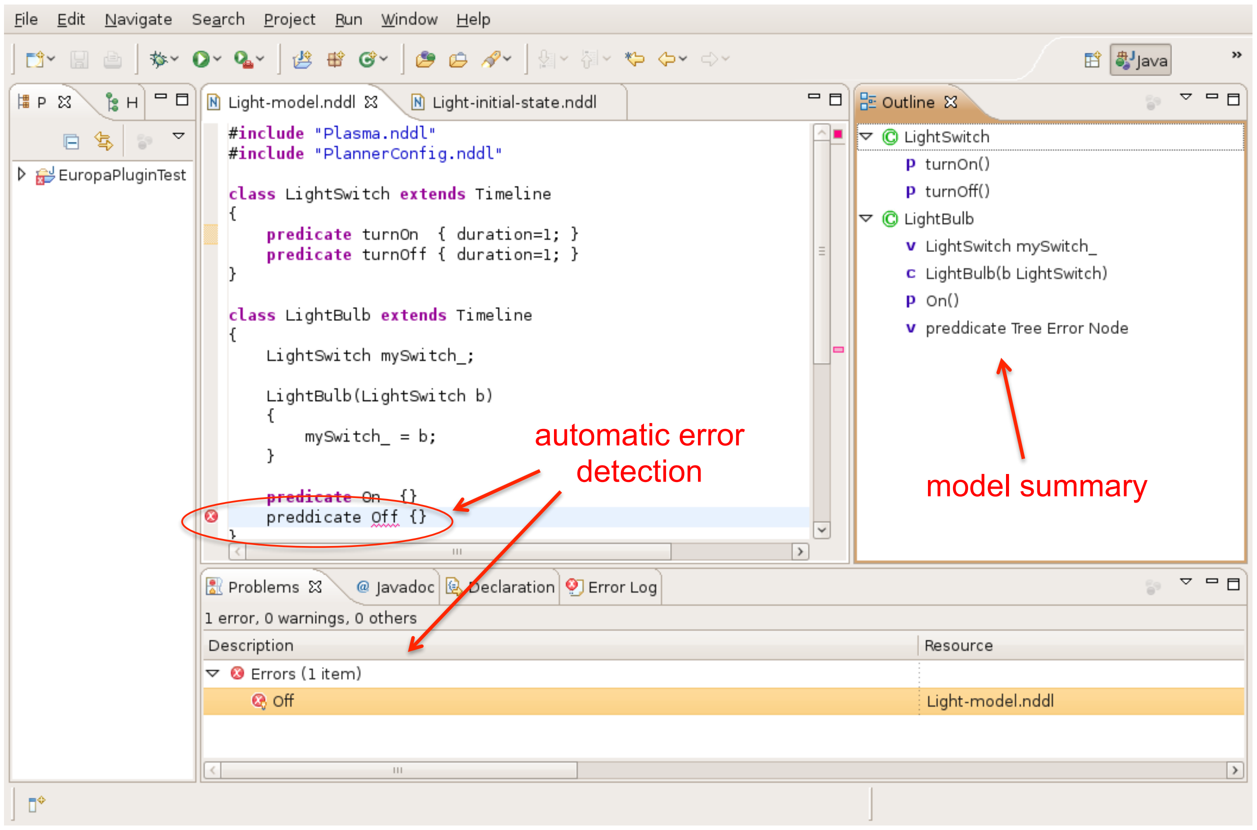
Using *EUROPA* & its Supporting Tools

There are several different ways in which *EUROPA* can be used to to support solving a planning & scheduling or CSP problem: (1) embed *EUROPA* within the client application; (2) using *PSDesktop*, a Java Swing UI Framework; and (3) using the provided Eclipse plugin. For the rest of this section, we will outline those three different approaches and provide some examples.

Embed *EUROPA* in an Application

EUROPA provides a script called *makeproject* that will generate C++ and Java applications that embed *EUROPA*, along with simple NDDL model and initial-state files that users can then modify for their own purposes. This allows the users to perform the full application cycle of:

1. Initialize *EUROPA*
2. Load/Modify model and initial state descriptions
3. Invoke a solver
4. Extract plan results from the Plan Database
5. Repeat steps 2-4 as many times as needed



```
#include "Plasma.nddl"
#include "PlannerConfig.nddl"

class LightSwitch extends Timeline
{
    predicate turnOn  { duration=1; }
    predicate turnOff { duration=1; }
}

class LightBulb extends Timeline
{
    LightSwitch mySwitch_;

    LightBulb(LightSwitch b)
    {
        mySwitch_ = b;
    }

    predicate On  {}
    preddicate Off {}
}
```

```
LightSwitch
    p turnOn()
    p turnOff()
LightBulb
    v LightSwitch mySwitch_
    c LightBulb(b LightSwitch)
    p On()
    v preddicate Tree Error Node
```

1 error, 0 warnings, 0 others

Description	Resource
Errors (1 item)	
Off	Light-model.nddl

Figure 2: NDDL Editor and Syntax Checker.

6. Shutdown *EUROPA*

JRB: Provide a short snippet showing a ”hello world” java app that embeds europa?

Using the C++ or JAVA API: The *PSEngine* interface is the official interface for *EUROPA* clients, it is the recommended way to use *EUROPA*. This interface is very straightforward and allows the user to run the entire application cycle described above. This abstraction layer will isolate a user's client code from most changes in the internals of the *EUROPA* implementation, it is also designed for easy mapping to other languages using SWIG.

Applications using different programming languages: the corresponding binding for the *PSEngine* interface should be either already available in the *EUROPA* distribution, or relatively easy to add. Note that while currently only Java bindings are bundled with the *EUROPA* distribution, we have plans to add Python and any other languages that are popular with the *EUROPA* user community [3].

JAVA Swing UI Framework

PSDesktop is a Java application that allows the user to drive *EUROPA* interactively by making use of the *PSEngine* client interface. It takes two arguments:

- To select *Debug* or *Optimized* version of *EUROPA* to run.
- *bsh* file (optional) : filename of the *BeanShell* file that is executed upon starting[4].

Figure **??** shows the two console windows when running *PSDesktop*. In the BeanShell console window you can type in Java statements that allow you to drive *EUROPA* interactively through its Java API. JRB: The NDDL console is currently broken! In the NDDL console you can type in NDDL statements that will be interpreted as soon as you complete a valid NDDL statement and hit enter.

Interactive BeanShell Console: In the BeanShell console, users will have access to the following variables :

- *PSEngine*: this will give you access to the *EUROPA* engine, users can create a solver, query the plan database, execute NDDL scripts, in general perform any task needed to drive *EUROPA* to load a model and create a plan. Users can also use this interface to create your own custom solver.
- *PSDesktop*: this will give you access to many utility methods to create new desktop windows, display tables of tokens, create a solver, etc.

PSUI Components: The PSUI package contains a number of components that make it easy to visualize your plan and interact with *EUROPA* :

- *PSGantt* : shows the tokens on a timeline as a gantt chart
- *PSChart* : shows resource profiles as charts
- *PSSolverDialog*: allows the user to drive a solver interactively and see its status as it tries to achieve the goals specified for it
- *ActionDetails and ActionViolation*: enable easy display of violation and detail information about actions in a plan as the user mouses over actions in other components (for instance a gantt chart)

Figure **??** shows an example of how different UI components within the PSUI package can be activated to assist the plan analysis.

Eclipse Plugin (SWT) for *EUROPA* Modeling, Solving, and Debugging

The *Eclipse* plugin has two major components: (1) an editor and (2) an execution perspective. They provide the graphical interface to model, run, and analyze plans within the *Eclipse* development environment. The main capabilities are:

- *NDDL Editor*: Syntax highlighting, syntax errors reporting, and linking structure to standard Outline View.
- *Solver View*: Start/stop the *EUROPA* engine, and configure and run a solver.
- *Statistics View*: Graphs of solver stats.
- *Open Decision View*: View of open decisions at each step of solving.
- *Schema Browser View*: View the schema for the active NDDL model.
- *Gantt View*: Once a solution is found, view the plan.
- *Details View*: Click on a token in the Gantt View to see it's details in this view.
- *Run NDDL model perspective*: Includes all of the above components.

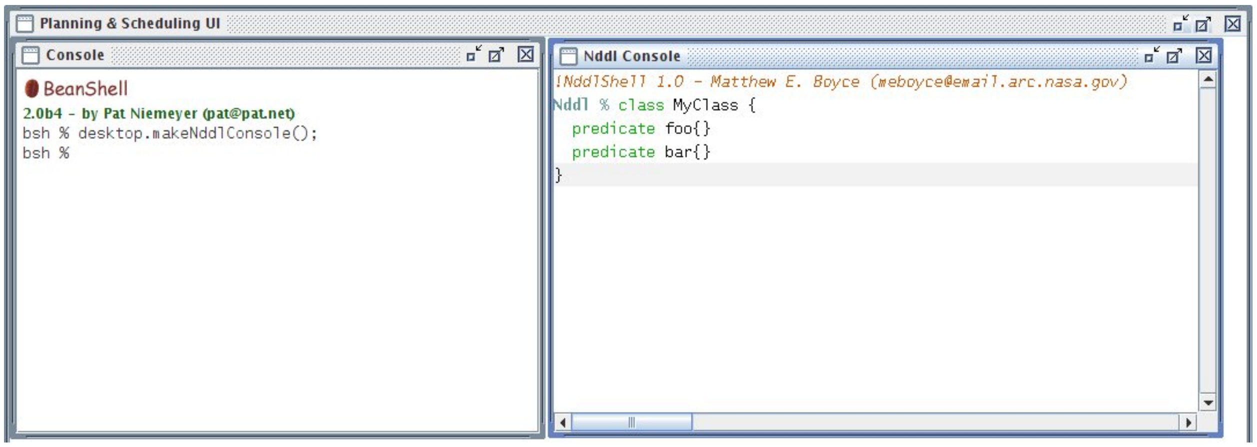
NDDL Graphical Model Editor: Eclipse plugin registers a file type for ".nddl" and a default editor for it. The editor has *syntax highlighting* and an *outline*, which is updated every time an editor is saved. If the parser detects any errors, they are displayed as error markers in the editor. Figure **??** shows the GUI for editing and checking the NDDL files.

Users can run *EUROPA* by directly invoking the *Run As* action for a given NDDL file. This action shows up both in the editor and in the *Package Explorer* pane. It creates a launch configuration and switches the perspective to NDDL model execution. Figure **??** depicts this option graphically. Figure **??** shows how to configure the run directly from within Eclipse.

Run NDDL Model Perspective: The Run as NDDL model perspective is the Eclipse version of the JAVA Swing *PSDesktop* user interface. The plugin can run multiple NDDL sessions at the same time. Users can switch between them using the pulldown list. *EUROPA* sessions are also visible in the Debug perspective and can be killed or restarted from there. Figure **??** shows an example of this perspective where

[3]This should not be too difficult given that SWIG supports many high-level programming languages.

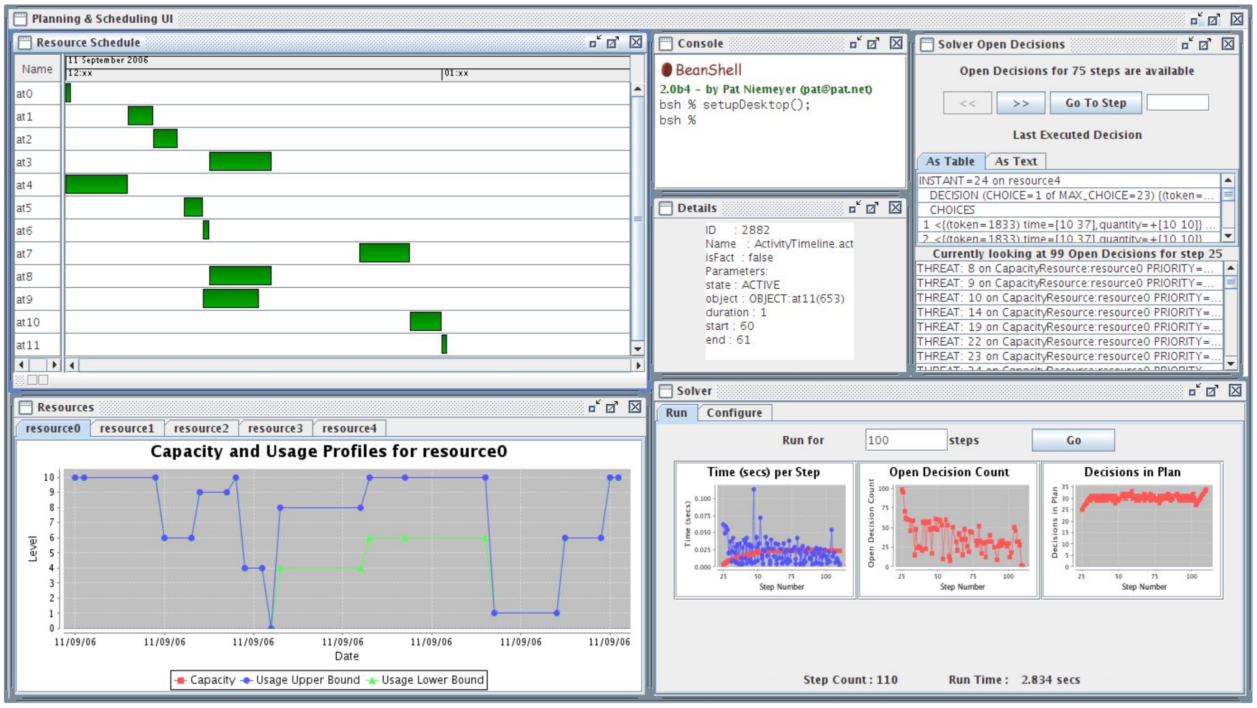
[4]BeanShell is a small, free, embeddable Java source interpreter with object scripting language features, written in Java. BeanShell dynamically executes standard Java syntax and extends it with common scripting conveniences such as loose types, commands, and method closures like those in Perl and JavaScript.



```
BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % desktop.makeNddlConsole();
bsh %
```

```
!NddlShell 1.0 - Matthew E. Boyce (meboyce@email.arc.nasa.gov)
Nddl % class MyClass {
  predicate foo{}
  predicate bar{}
}
```

Figure 3: BeanShell and NDDL console windows



```
BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % setupDesktop();
bsh %
```

```
ID     : 2882
Name   : ActivityTimeline.act
isFact : false
Parameters:
state : ACTIVE
object : OBJECT:at11(653)
duration : 1
start : 60
end : 61
```

Open Decisions for 75 steps are available

```
INSTANT=24 on resource4
  DECISION (CHOICE=1 of MAX_CHOICE=23) {(token=...
  CHOICES
 1 <{(token=1833) time=[10 37],quantity=+[10 10]} ...
 2 <{(token=1833) time=[10 37],quantity=+[10 10]}
```

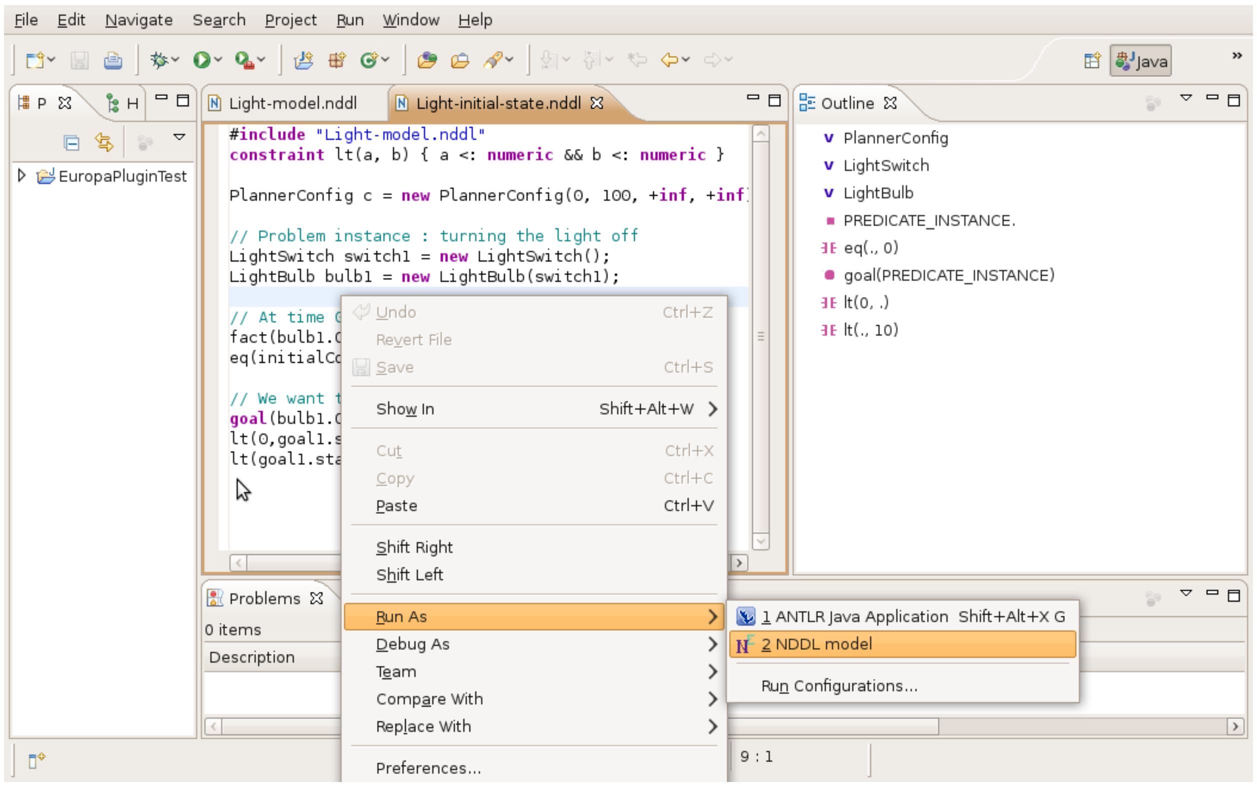
Currently looking at 99 Open Decisions for step 25

```
THREAT: 8 on CapacityResource:resource0 PRIORITY=...
THREAT: 9 on CapacityResource:resource0 PRIORITY=...
THREAT: 10 on CapacityResource:resource0 PRIORITY=...
THREAT: 14 on CapacityResource:resource0 PRIORITY=...
THREAT: 19 on CapacityResource:resource0 PRIORITY=...
THREAT: 22 on CapacityResource:resource0 PRIORITY=...
THREAT: 23 on CapacityResource:resource0 PRIORITY=...
```

Run for 100 steps

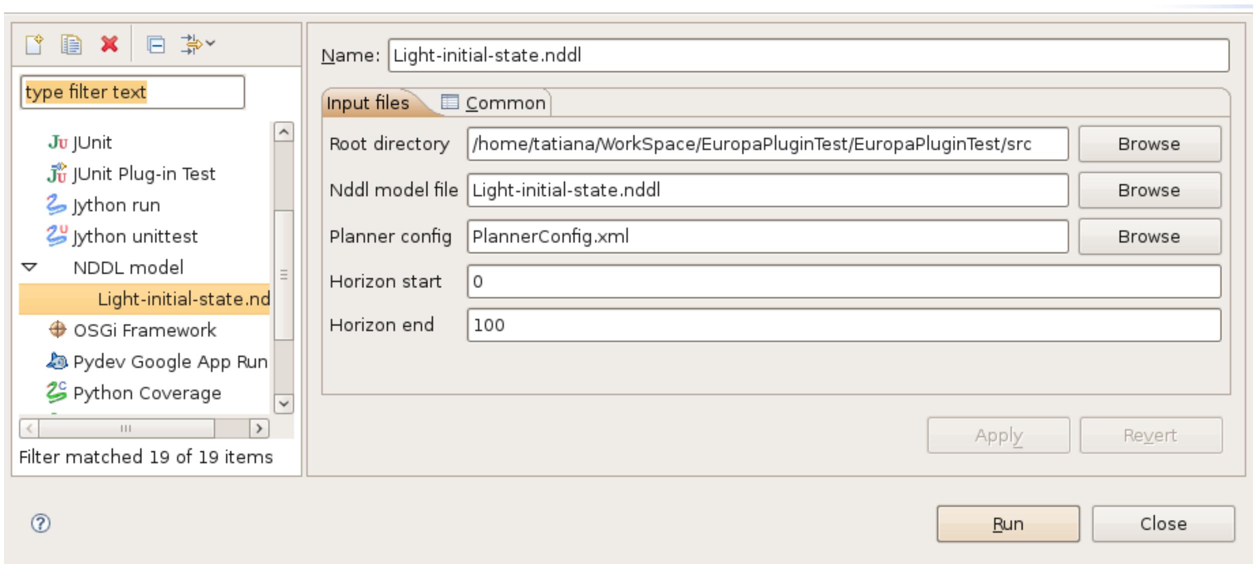
Step Count : 110   Run Time : 2.834 secs

Figure 4: Example of PSUI components



```
#include "Light-model.nddl"
constraint lt(a, b) { a <: numeric && b <: numeric }

PlannerConfig c = new PlannerConfig(0, 100, +inf, +inf

// Problem instance : turning the light off
LightSwitch switch1 = new LightSwitch();
LightBulb bulb1 = new LightBulb(switch1);

// At time 0
fact(bulb1.0
eq(initialCo

// We want t
goal(bulb1.0
lt(0,goal1.s
lt(goal1.sta
```

Figure 5: Running a problem directly from the NDDL editor.



Name: Light-initial-state.nddl

Root directory: /home/tatiana/WorkSpace/EuropaPluginTest/EuropaPluginTest/src

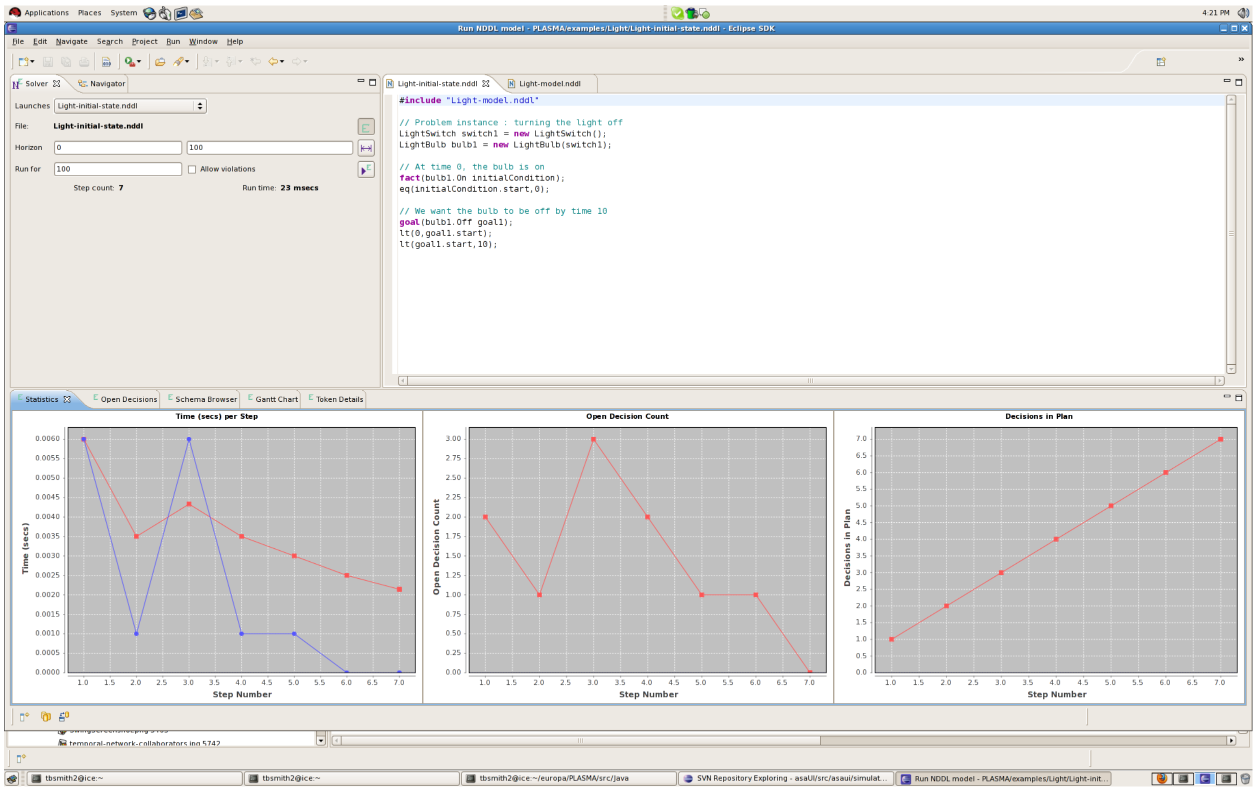
Nddl model file: Light-initial-state.nddl

Planner config: PlannerConfig.xml

Horizon start: 0

Horizon end: 100

Figure 6: Configurate a NDDL run.



Launches: Light-initial-state.nddl
File: **Light-initial-state.nddl**
Horizon: 0 100
Run for: 100 Allow violations
Step count: **7** Run time: **23 msecs**

```
#include "Light-model.nddl"

// Problem instance : turning the light off
LightSwitch switch1 = new LightSwitch();
LightBulb bulb1 = new LightBulb(switch1);

// At time 0, the bulb is on
fact(bulb1.On initialCondition);
eq(initialCondition.start,0);

// We want the bulb to be off by time 10
goal(bulb1.Off goal1);
lt(0,goal1.start);
lt(goal1.start,10);
```

Figure 7: Eclipse dedicated perspective on showing information about running *EUROPA*

different aspects of modeling and execution can be visually displayed.

Examples

In this section, we show several examples that demonstrate the flexibility of *EUROPA* (both its core engine and its supporting knowledge engineering tools) when solving different types of planning, scheduling, and constraint satisfaction problems. All examples covered in this section are included in the *EUROPA* distribution and in this section they are illustrated through the *PSDesktop* interface.

Light: A "Hello World" domain for *EUROPA* , it describes how a light switch can be used to control a light bulb. Figure **??** shows an example output analyzing the final plans. In this particular example, the intervals mean that the action or state change could happen at any point in the interval, so for instance, "*lightSwitch1 is turned off at time [0,8]*" means that *lightSwitch1* could be turned off at time 0, or at time 1, ..., or at time 8. You can modify your model or *EUROPA*'s configuration to generate grounded plans (where all the values are points, instead of intervals), if that's what you want for your particular application.

N-Queens: N-Queens is one of CSP's workhorses. Figure **??** shows how *EUROPA* supports model and solve this problem through *PSDesktop*. Users can click on a chess board to move the the queens around and see the constraint violations that *EUROPA* computes by moving the mouse over each queen. It also provides a simple Tabu Search solver which briefly illustrates how users can build their own solvers on top of *EUROPA*.

Resource Constrained Project Scheduling Problem (RCPSP): this is a well known problem in the OR community that consists of scheduling a set of activities with temporal and resource constraints. Typically, the goal is to minimize total project duration while respecting all constraints (Figure **??**). Like previous example, this example shows how users can build their own solvers on top of *EUROPA* for a specific problem.

Shopping: a simple example discussed in Russel and Norvig's AI textbook, first Edition, Chapter 11 on Planning (Figure **??**).

Blocksworld: this is one of the most well-known planning domains. This version uses a robotic arm to build the stacks and Figure **??** shows a UI where you can look at the partial state of the arm and the stacks as the planner progresses towards the stated goal. The *PSDesktop* UI allows users to mouse over the green rectangles to see the actions over each timeline; for example, you can see the arm operator performing pick up and stack operations on the blocks. The "BlockWorld History" window shows the evolution of the stacks as the operator performs the actions from the plan on them, until it arrives to the stated goal.

Planetary Rover: this is a more complex planning domain that was inspired by NASA robotic missions and has also been translated to PDDL to be used in recent IPCs. Figure **??** shows the UI result of this domain. In this figure, red and blue curves on the chart at the top-right corner bound the possible battery charge. The difference between the two is due to flexibility in the plan regarding when navigation and sampling may take place. The red curve shows the charge when they occur as soon as possible, and the blue curve shows them if everything is delayed as long as possible. The bottom window displays a gantt chart for the Rover, Navigator and Instrument timelines in this problem. Hover the mouse over any piece (green rectangle) of the gantt chart to see details displayed in the Details window. In this screenshot, the mouse was hovered over the large box on the Navigator timeline, which is an *At* predicate.

EUROPA-related Projects

EUROPA has been used for a variety of missions, mission-oriented research, and demonstrations, including:

JRB: I took an initial pass at this list, dropped some where europa's involvement was non material in the end. I also tried to reorganize going from production apps to prototypes. Need to make another pass

- DS1: RAX Remote Agent Experiment (original version of technology)
- SACE Support for operation of the International Space Station's solar arrays
- Bedrest study at Johnson Space Center
- MER Tactical Activity Planning. *EUROPA* is the core planning technology behind MAPGEN, a decision support tool for generating detailed activity plans on a daily basis for the MER robotic mission to Mars.
- Mars 03: MER Mars Exploration Rover Science Operations
- MSL : Support for planning and scheduling for Mars Science Laboratory Science Operations
- Intelligent Distributed Execution Architecture (IDEA)
- On-board Planning and Plan Execution. *EUROPA* was the core planning technolgoy for deliberative and reactive planning on-board a variety of mobile robots. It has been fielded in the Atacama Desert and was the cornerstone of a 2005 milestone of human-robotic collaboration for the Collaborative Decision Systems program.
- Crew Planning Research project on Planning and Scheduling for space missions
- ATHLETE support for foot fall planning for a lunar robot
- STAR Advanced Spaceflight Training Systems Development
- Mission Simulation. *EUROPA* was used to simulate a prospective robotic mission (LORAX) to the Antarctic for the purposes of system design evaluation.
- Contingent Planning for ROVER operations (PiCO)
- Personal Satellite Assistant (PSA)
- Spoken Interface Prototype for PSA (RIALIST)
- IS Milestone
- CDS Milestone

Outside of NASA, it has also beed used at MBARI to help control underwater autonomous vehicle (**?**) and at Willow Garage for autonomous robot navigation (**?**).

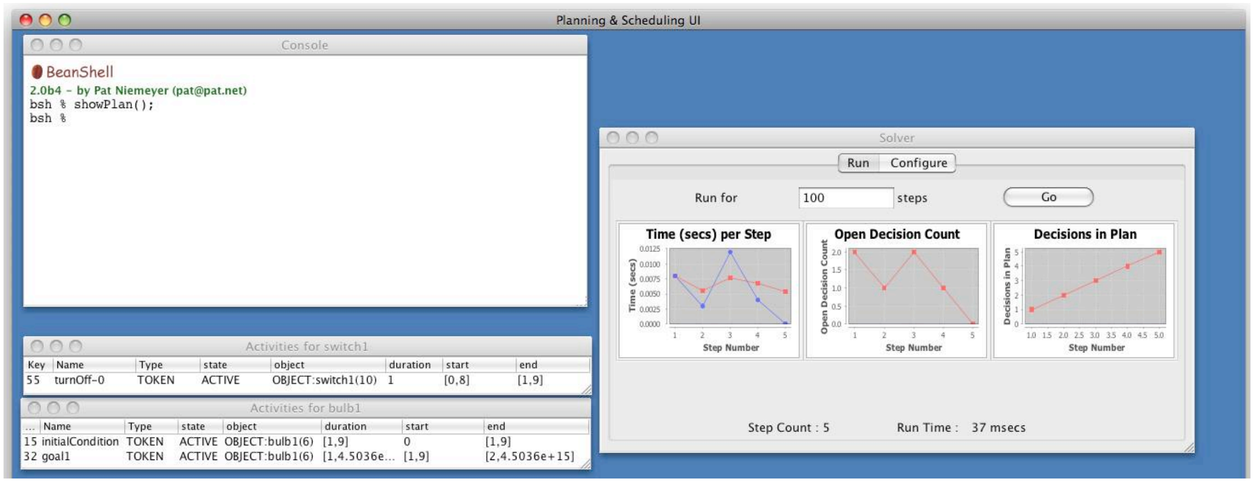
Conclusion and Future Work

In this paper, we described *EUROPA* with concentration on its modeling and plan analysis capabilities. The main strengths of *EUROPA* are: (1) expressive; (2) flexible framework; (3) strong support for integration with other applications; (4) open-source license; and (5) proven track record.

While *EUROPA* and its supporting tools have been going through a long period of development, we still have a long list of improvements that we want to make. The most important ones in our opinion are: significantly improve search (especially heuristic guidance) and inference capabilities, support the ANML and PDDL modeling languages, improve the visualization and debugging tools and allow *EUROPA* extensions to be written in other languages. Given that *EUROPA* is open-source software, we welcome contributions from planning and scheduling researchers and practitioners.

Acknowledgements: *EUROPA* is the result of many years of research, development and deployment of constraint-based planning technology.

- The precursor to *EUROPA* was HSTS, designed and developed by Nicola Muscettola. HSTS set out the initial domain description language and essentials of the planning paradigm that became the starting point for *EUROPA*.
- Ari Jonsson led the implemenation of the first version of *EUROPA*Ȧri's team included Jeremy Frank, Paul Morris and Will Edgington, who all made valuable contributions.
- Conor McGann led the implementation of *EUROPA* 2, which is a further evolution of this line of work, targeted mainly at making the technology easier to use, more efficient, easier to integrate and easier to extend. *EUROPA* 2's main contributors were Andrew Bachmann, Tania Bedrax-Weiss, Matthew Boyce, Patrick Daley, Will Edgington, Jeremy Frank, Michael Iatauro, Peter Jarvis, Ari Jonsson, Paul Morris, Sailesh Ramakrishnan and Will Taylor.
- Javier Barreiro took over as the *EUROPA* team lead in the Fall of 2006 and has been working on it since then, improving *EUROPA*'s technology and packaging. Javier's main collaborators at NASA Ames are Matthew Boyce, Minh Do, Michael Iatauro, Paul Morris, Tristan Smith and David Smith.



```
BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % showPlan();
bsh %
```

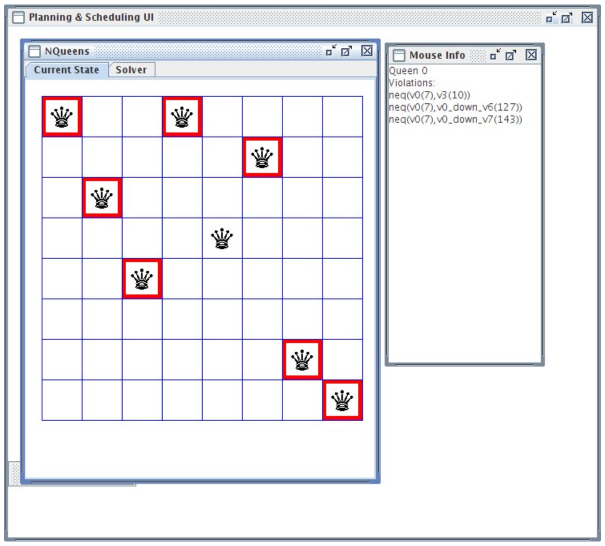
Activities for switch1

Key	Name	Type	state	object	duration	start	end
55	turnOff-0	TOKEN	ACTIVE	OBJECT:switch1(10)	1	[0,8]	[1,9]

Activities for bulb1

...	Name	Type	state	object	duration	start	end
15	initialCondition	TOKEN	ACTIVE	OBJECT:bulb1(6)	[1,9]	0	[1,9]
32	goal1	TOKEN	ACTIVE	OBJECT:bulb1(6)	[1,4.5036e...	[1,9]	[2,4.5036e+15]

Figure 8: UI example of the simple light-switch domain where the only action is to turn a light *ON* or *OFF*.



```
Queen 0
Violations:
neq(v0(7),v3(10))
neq(v0(7),v0_down_v6(127))
neq(v0(7),v0_down_v7(143))
```

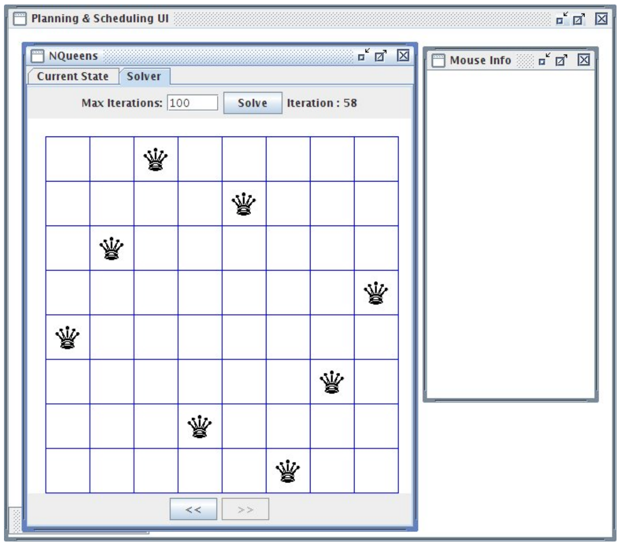
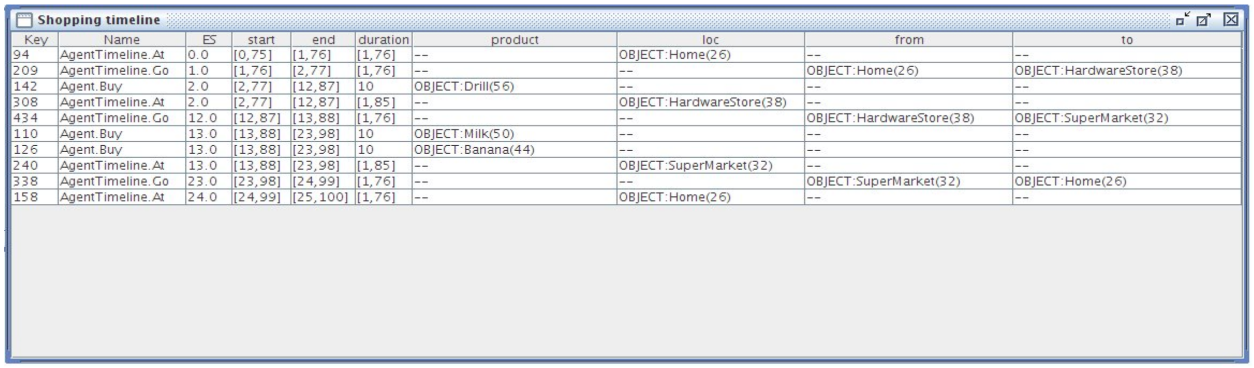
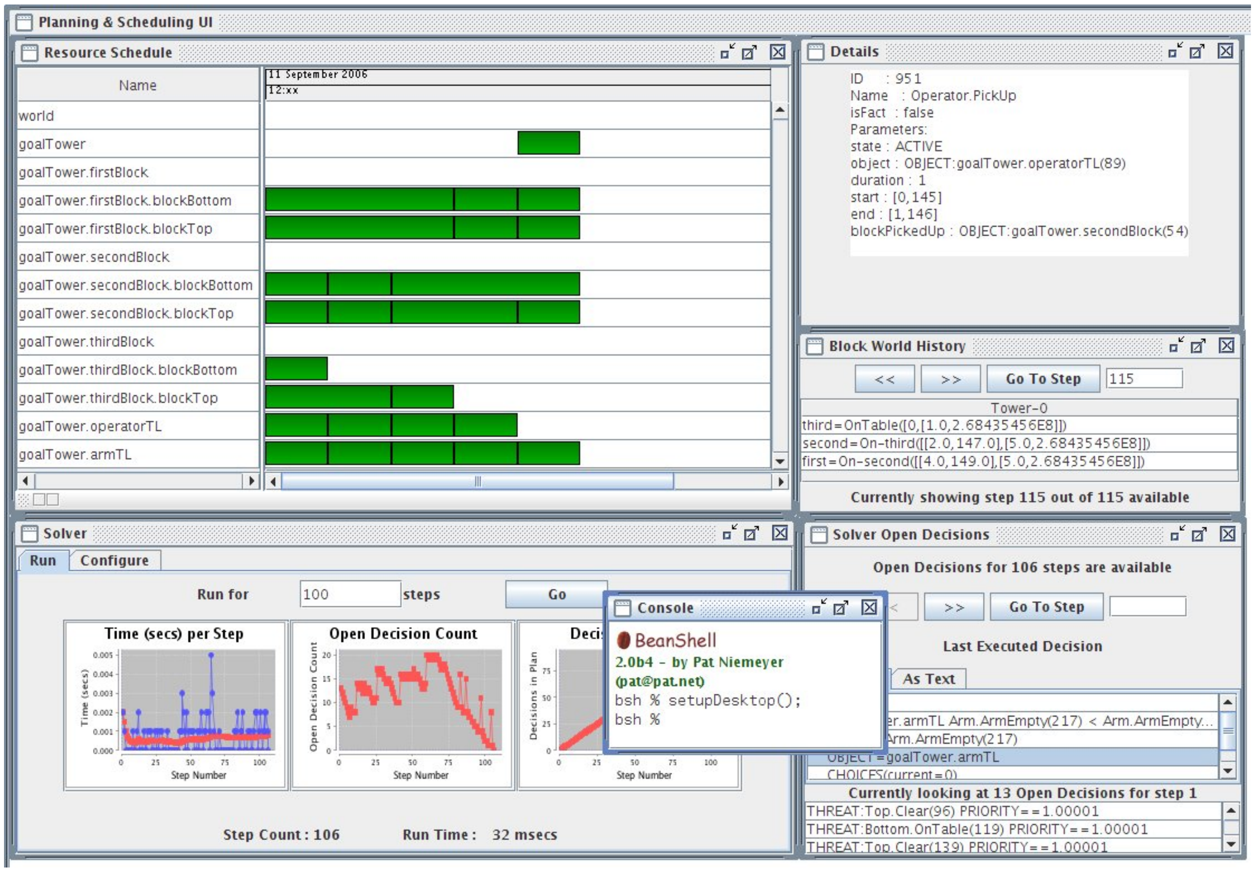


Figure 9: UI example of the representative Constraint Programming domain: NQueens


Shopping timeline

Key	Name	ES	start	end	duration	product	loc	from	to
94	AgentTimeline.At	0.0	[0,75]	[1,76]	[1,76]	--	OBJECT:Home(26)	--	--
209	AgentTimeline.Go	1.0	[1,76]	[2,77]	[1,76]	--	--	OBJECT:Home(26)	OBJECT:HardwareStore(38)
142	Agent.Buy	2.0	[2,77]	[12,87]	10	OBJECT:Drill(56)	--	--	--
308	AgentTimeline.At	2.0	[2,77]	[12,87]	[1,85]	--	OBJECT:HardwareStore(38)	--	--
434	AgentTimeline.Go	12.0	[12,87]	[13,88]	[1,76]	--	--	OBJECT:HardwareStore(38)	OBJECT:SuperMarket(32)
110	Agent.Buy	13.0	[13,88]	[23,98]	10	OBJECT:Milk(50)	--	--	--
126	Agent.Buy	13.0	[13,88]	[23,98]	10	OBJECT:Banana(44)	--	--	--
240	AgentTimeline.At	13.0	[13,88]	[23,98]	[1,85]	--	OBJECT:SuperMarket(32)	--	--
338	AgentTimeline.Go	23.0	[23,98]	[24,99]	[1,76]	--	--	OBJECT:SuperMarket(32)	OBJECT:Home(26)
158	AgentTimeline.At	24.0	[24,99]	[25,100]	[1,76]	--	OBJECT:Home(26)	--	--

Figure 10: UI example of the text-book *shopping* planning example



Name	11 September 2006 12:xx
world	
goalTower	🟩
goalTower.firstBlock	
goalTower.firstBlock.blockBottom	🟩🟩🟩
goalTower.firstBlock.blockTop	🟩🟩🟩
goalTower.secondBlock	
goalTower.secondBlock.blockBottom	🟩🟩🟩
goalTower.secondBlock.blockTop	🟩🟩🟩🟩
goalTower.thirdBlock	
goalTower.thirdBlock.blockBottom	🟩
goalTower.thirdBlock.blockTop	🟩🟩
goalTower.operatorTL	🟩🟩🟩🟩
goalTower.armTL	🟩🟩🟩🟩🟩

```
ID     : 951
Name   : Operator.PickUp
isFact : false
Parameters:
state : ACTIVE
object : OBJECT:goalTower.operatorTL(89)
duration : 1
start : [0,145]
end : [1,146]
blockPickedUp : OBJECT:goalTower.secondBlock(54)
```

115

Tower-0
third=OnTable([0,[1.0,2.68435456E8]])
second=On-third([[2.0,147.0],[5.0,2.68435456E8]])
first=On-second([[4.0,149.0],[5.0,2.68435456E8]])

Currently showing step 115 out of 115 available

100

Step Count : 106   Run Time : 32 msecs

```
BeanShell
2.0b4 - by Pat Niemeyer
(pat@pat.net)
bsh % setupDesktop();
bsh %
```

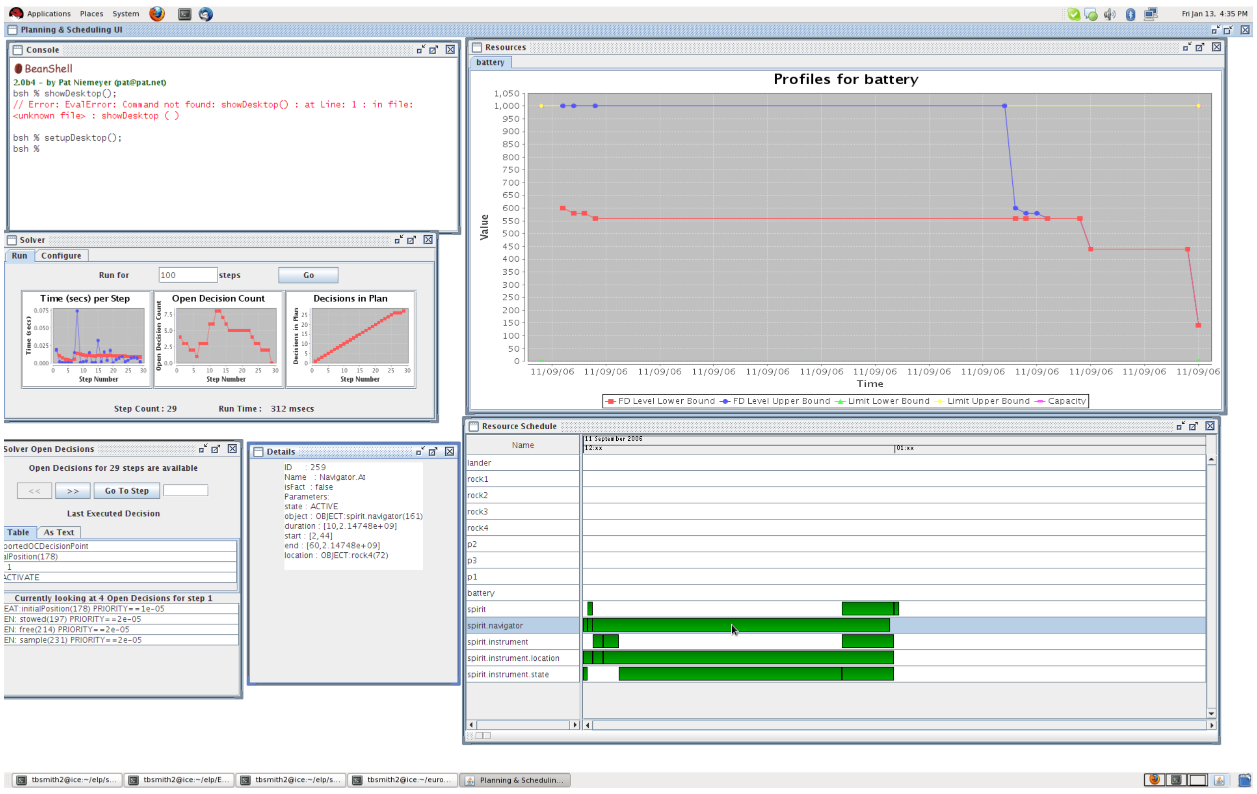
Open Decisions for 106 steps are available

```
er.armTL Arm.ArmEmpty(217) < Arm.ArmEmpty...
Arm.ArmEmpty(217)
OBJECT=goalTower.armTL
CHOICES(current=0)
```

Currently looking at 13 Open Decisions for step 1

```
THREAT:Top.Clear(96) PRIORITY==1.00001
THREAT:Bottom.OnTable(119) PRIORITY==1.00001
THREAT:Top.Clear(139) PRIORITY==1.00001
```

Figure 11: UI example of the classical *Blocksworld* planning domain.



```
BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % showDesktop();
// Error: EvalError: Command not found: showDesktop() : at Line: 1 : in file:
<unknown file> : showDesktop ( )

bsh % setupDesktop();
bsh %
```

Run for 100 steps

Step Count : 29   Run Time : 312 msecs

Open Decisions for 29 steps are available

Last Executed Decision

```
oortedOCDecisionPoint
alPosition(178)
1
ACTIVATE
```

Currently looking at 4 Open Decisions for step 1

```
EAT:initialPosition(178) PRIORITY==1e-05
EN: stowed(197) PRIORITY==2e-05
EN: free(214) PRIORITY==2e-05
EN: sample(231) PRIORITY==2e-05
```

```
ID      : 259
Name    : Navigator.At
isFact  : false
Parameters:
state : ACTIVE
object : OBJECT:spirit.navigator(161)
duration : [10,2.14748e+09]
start : [2,44]
end : [60,2.14748e+09]
location : OBJECT:rock4(72)
```

Figure 12: UI example of the *Rovers* planning domain.

External contributors and collaborators include: Tatiana Kichkaylo, Mark Roberts, and Tony Pratkanis. Funding for this work has been provided by the NASA Intelligent Systems and Collaborative Decision Systems Programs.

References

Frank, J., and Jonsson, A. K. 2003. Constraint-based attribute and interval planning. *Journal of Constraints Special Issue on Constraints and Planning* 8(4).

Gerevini, A.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence* 173:619–668.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of AIPS-94*, 61–67.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for auv control. In *Proc. of Intnl. Conf. on Robotics and Automation (ICRA)*.

McGann, C.; Berger, E.; Bohren, J.; Chitta, S.; Gerkey, B.; Glaser, S.; Marthi, B.; Meeussen, W.; Pratkanis, T.; Marder-Eppstein, E.; and Wise, M. 2009. Model-based, hierarchical control of a mobile manipulation platform. In *Proc. of ICAPS Workshop on Planning and Plan Execution for Real-World Systems*.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103:5–47.

Smith, D. E., and Bernardini, S. 2010. Translating pddl into nddl.

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *Proceedings of ICAPS-08*.