EUROPA 2: Plan Database Services for Planning and Scheduling Applications

Tutorial

Tania Bedrax-Weiss (PI)

Andrew Bachman, Patrick Daley (intern), Will Edgington, Michael Iatauro, Conor McGann, Will Taylor (PlanWorks)

Objectives

To Understand:

- Constraint-Based Planning Paradigm
- EUROPA 2 and its Use Cases
- How to Create Your Own Project
- How to Generate a Plan and Visualize it in PlanWorks
- Possible Extensions and Create Your Own Constraint
- Modeling Features and Their Use

Overview

Part I

- Motivation
- Background on Constraint-Based Planning

Part II

- Constraint Engine
- Plan Database
- NDDL
- Assemblies
- PlanWorks
- Aver
- Extensions

Part III

- Build your own model
- Visualize it in PlanWorks

Motivation

Needs

- Describe a large class of problems relevant to space exploration
- Implement a wide variety of planning algorithms
- Perform advanced inference automatically
- Provide services to a wide variety of applications/architectures

Technology Components

- A powerful modeling language – describes the problem domain
- A robust and powerful Plan Database – enforces plan consistency inferring consequences of client transactions
- A Planner Application Framework – straw-man for building a planner

Benefits

- Increase capabilities of mission and research applications
- Reduce development cost and risk
- Encourage technology transfer through common infrastructure

Applications

Missions

- DS1: RAX – Remote Agent Experiment (original version of technology)
- Mars ‘03: MER – Mars Exploration Rover Science Operations
- Mars ‘09? : MSL – Mars Science Laboratory Science Operations

Mission-oriented research

- Intelligent Distributed Execution Architecture (IDEA – EUROPA, EUROPA 2)
- Earth-observing satellite scheduling project (EOS – EUROPA, EUROPA 2)
- SOFIA flight scheduling project (SOFIA – EUROPA)
- Contingent Planning for ROVER operations (PiCO – EUROPA 2)
- Personal Satellite Assistant (PSA – EUROPA)
- Spoken Interface Prototype for PSA (RIALIST – EUROPA)

Demonstrations

- IS Milestone (EUROPA 2, support ended in 2004)
- CDS Milestone (EUROPA 2, currently supporting)

Research

- Preferences work (EUROPA 2)
- Mixed Initiative Tactical Planning (EUROPA 2, exploratory)

Requirements

Driven by domain needs:

- Concurrent operations with temporal dependencies
  - Instruments, mobility, communications, etc.
- Limited resource availability
  - Power, data storage, etc.
- Complex rules for interactions between operations
  - Example: Instruments require heating, interact with communications and mobility operations

Additional considerations:

- Efficiency and power of constraint reasoning
  - Temporal reasoning, resource reasoning, activity scheduling
- Support for different use cases
  - Fully automated planning, mixed-initiative, multi-agent planning, etc.
  - Flexibility in plan completeness criteria and generated plans

Automated Planning

Given: Partial plan, including desired goals

Process: Automatically modify candidate plan

Result: A complete valid plan, or inability to find one

Planner::step(P)

determine consequences of decisions in P

if P cannot lead to a valid plan, **return** failure

if P is a complete valid plan, **return** success

Planner selects decision x in P

Planner makes decision x

return step(P + x)

Mixed-Initiative Planning

Human and automated planner collaborate to produce plan

Process: User and automated system modify plan

- User makes decisions - automated system handles ramifications
- User requests help with decisions - automated system put to work
- Automated system decision overridden by user

Planner::step(P)

Planner determines consequences of decisions in P

if P cannot lead to a valid plan, **return** failure

if P is a complete valid plan, **return** success

User or planner selects decision x in P

User or planner makes decision x

return step(P + x)

Multi-Agent Planning

Multiple planners act on different views of the same plan

- Different temporal horizons
- Different objects, timelines, resources

Process: each agent modifies plan

- Each agent makes decisions - system handles ramifications
- Each agent requests help with decisions - system put to work

Planner::step(P)

Planner determines consequences of decisions in P

if P cannot lead to a valid plan, **return** failure

if P is a complete valid plan, **return** success

Agent selects decision x in **view** on P

Agent makes decision x

return step(P + x)

Consequences for CBP

Support for

- Automated Planning
- Mixed-Initiative Planning
- Multi-Agent Planning

Separate planning process from plan maintenance

Plan Database maintains plans and provides

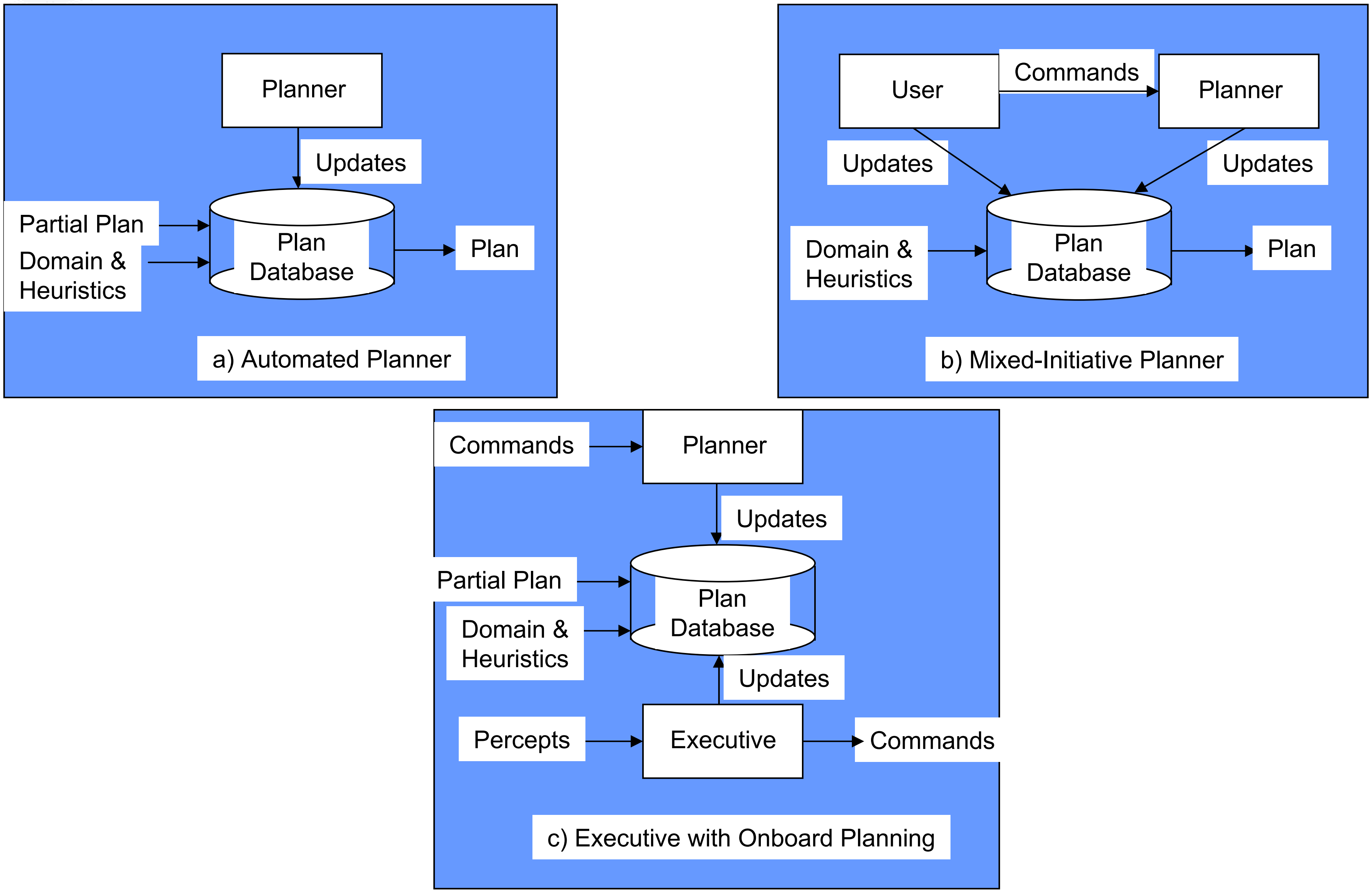
- Resource consistency services
- Temporal consistency services
- Constraint consistency services
- Subgoaling services

Planner maintains decisions and provides

- View into pending decisions
- Search control

Plan Database determines consequences of decisions made by planner.

Sample Architectures



Overview

Part I

- ✓ Motivation
- ❑ Background on Constraint-Based Planning

Part II

- ❑ Constraint Engine
- ❑ Plan Database
- ❑ NDDL
- ❑ Assemblies
- ❑ PlanWorks
- ❑ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Background on Constraint Based Planning

- Constraint Satisfaction Reasoning
- Simple Temporal Network Reasoning
- Procedural Constraint Reasoning
- Resource Reasoning
- Mapping Planning into Dynamic Constraint Satisfaction
  - Finite State Machine Model
  - Subgoaling
- Example

Many thanks to Ari Jónsson for providing many of the slides.

Constraint Satisfaction Problem

Problem defined by

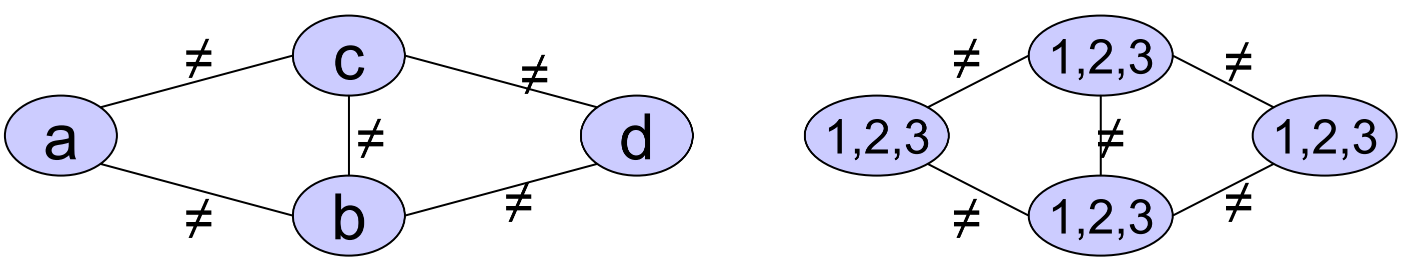
- Set of variables, each with a finite domain
- Set of constraints, restricting combinations of values

Solution to constraint problem

- Each variable assigned value from its domain
- All constraints are satisfied

Simple example

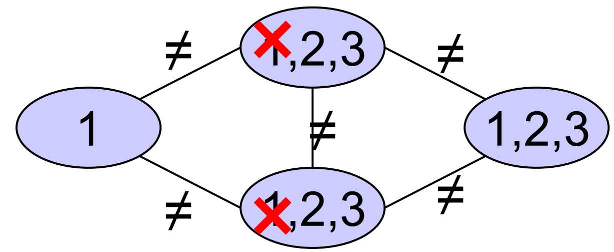
- Variables: a,b,c,d take values from domain {1,2,3}
- Constraints:



Constraint Reasoning

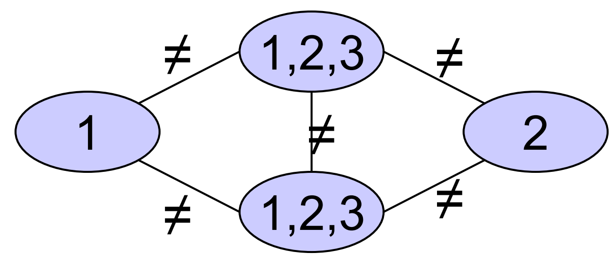
Eliminate impossible values:

- Value is eliminated if it cannot appear in any solution



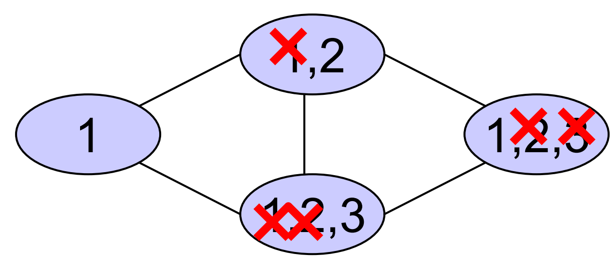
Determine consistency:

- Problem is consistent if a solution exists, inconsistent otherwise



Find solution:

- Find values satisfying constraints

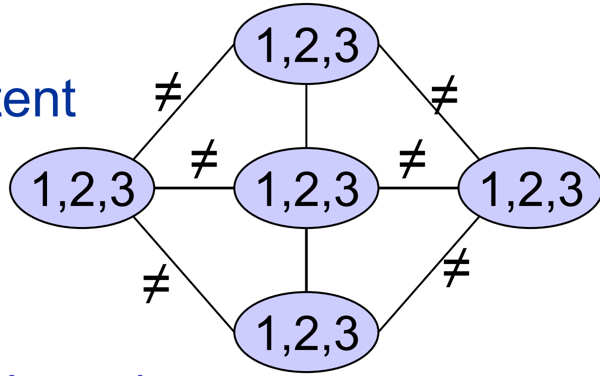


Arc Consistency

Binary constraint is arc-consistent if for every value in one variable there is satisfying value for other variable

Ternary constraint is 3-consistent if for every combination of three variables, assigning two of them allows assignment for third

CSP is arc-consistent if each constraint is arc-consistent



Achieving arc-consistency

- If a domain becomes empty, return inconsistent
- eliminate values for which no matching satisfying value exists
- repeat to quiescence

Dynamic Constraint Problems

Constraint problems as part of larger problem

- Constraint-based planning
- Design synthesis
- Automated diagnosis

Constraint problems change over time

- Variables and constraints are added and deleted
- Elements of domains are added and deleted

Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ❑ Simple Temporal Network Reasoning
- ❑ Procedural Constraint Reasoning
- ❑ Resource Reasoning
- ❑ Mapping Planning into Dynamic Constraint Satisfaction
  - ❑ Finite State Machine Model
  - ❑ Subgoaling
- ❑ Example

Simple Temporal Reasoning

Temporal constraint network

- Variables represent event times
- Constraints relate event times

Simple temporal network

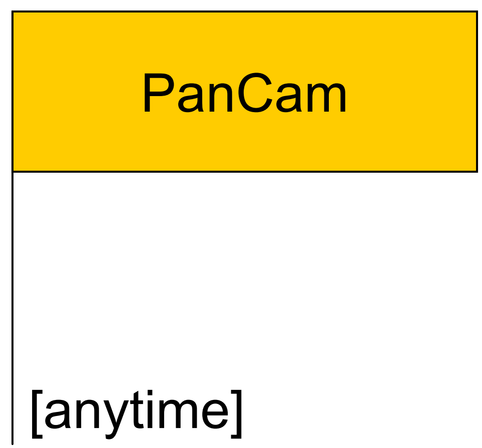
- Domain of each variable is a temporal interval
- Constraints specify distance bounds on variable pairs

Efficient reasoning for simple temporal networks

- Consistency can be determined in polynomial time

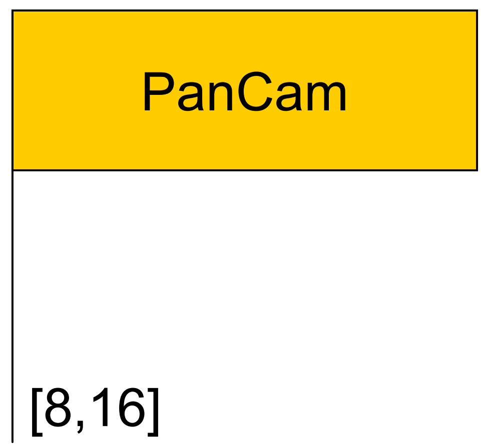
Temporal Flexibility Example

Initially



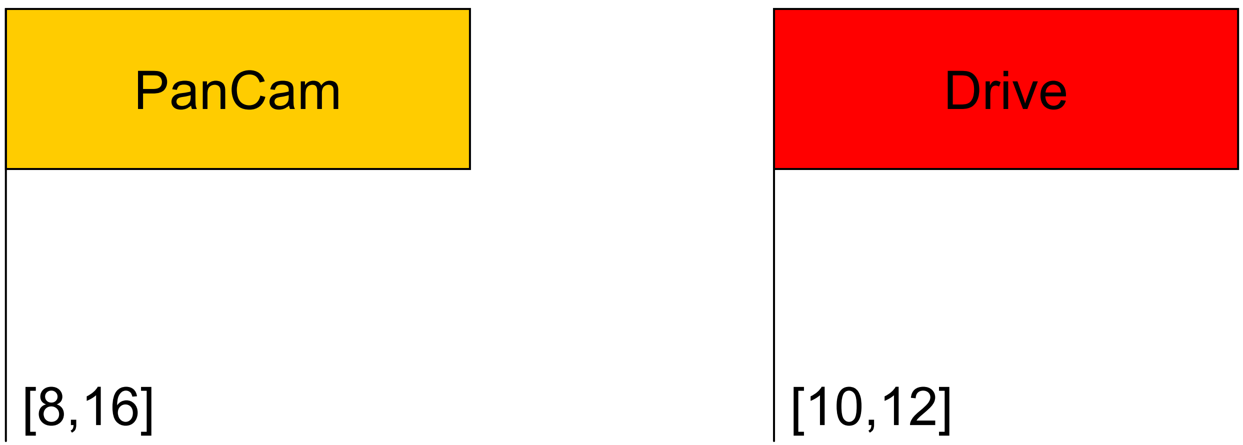
Example

Add constraint: PanCam starts between 8 and 16



Example

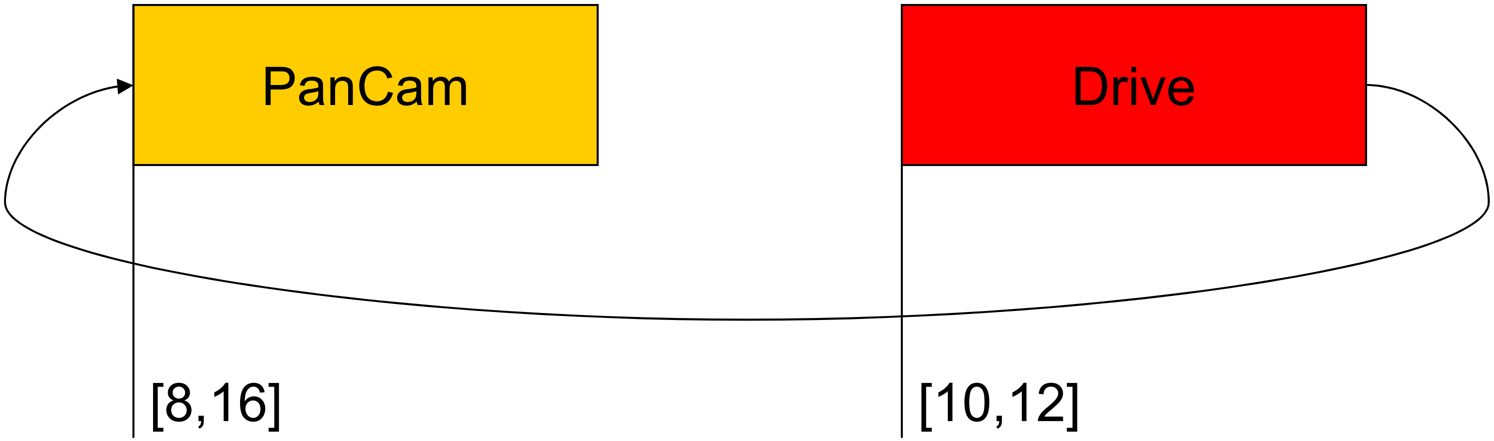
Constraint 1: PanCam starts between 8 and 16
Constraint 2: Drive starts between 10 and 12



Example

Constraint 1: PanCam starts between 8 and 16
Constraint 2: Drive starts between 10 and 12

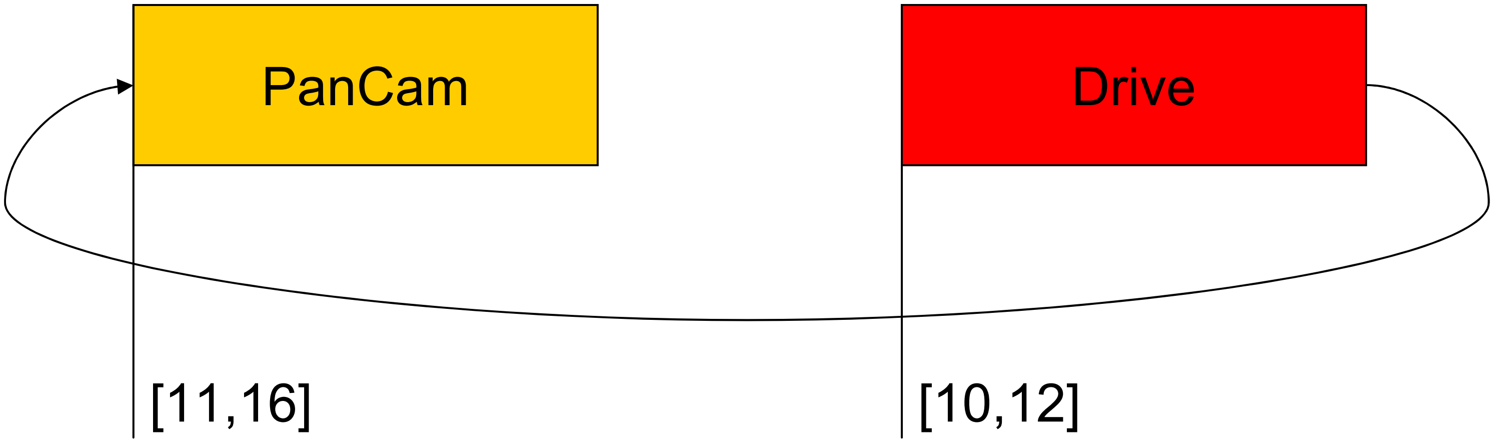
Add constraint: Start of PanCam after end of drive



Example

Constraint 1: PanCam starts between 8 and 16
Constraint 2: Drive starts between 10 and 12

Add constraint: Start of PanCam after end of drive
Impact: Reduces time range for PanCam



Duration is at least 1

Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ✓ Simple Temporal Network Reasoning
- ❑ Procedural Constraint Reasoning
- ❑ Resource Reasoning
- ❑ Mapping Planning into Dynamic Constraint Satisfaction
  - ❑ Finite State Machine Model
  - ❑ Subgoaling
- ❑ Example

Procedural Constraints

Exploiting constraint reasoning inefficiencies

- Comparison
- Arithmetic
- Differential Equations

Dynamic constraint reasoning using procedures

- Procedures replace (some) declarative constraints
- Allow variables and values to be dynamic
- Support real-valued reasoning (floats)
- Emphasis on propagation, not search (planner does search)

Constraint Reasoning for Planning

Representation

- Variables: objects and activity parameters, temporal information
- Constraints: parameter relations, Allen relations, constant relations

Reasoning

- Identify when plan candidate is inconsistent
- Eliminate choices leading to invalid plans

Requirements

- <u>General</u>: arbitrary constraints (domain-dependent)
- <u>Dynamic</u>: constraints, variables and values added/deleted
- <u>Efficient</u>: network changed and queried at each plan step
  - Trade-off between efficiency and completeness of reasoning

Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ✓ Simple Temporal Network Reasoning
- ✓ Procedural Constraint Reasoning
- ❑ Resource Reasoning
- ❑ Mapping Planning into Dynamic Constraint Satisfaction
  - ❑ Finite State Machine Model
  - ❑ Subgoaling
- ❑ Example

Resource reasoning

Bounding resource usage

- Flexible candidate plans give rise to bounds on resource use
- Need to calculate tight bounds to identify resource problems early, and provide guidance to search engine

Using external resource calculations

- In a current application, resource calculations provided by external simulation software
- Simulation only provides earliest start time resource profile
- Adapt search to reason with provided profiles

Combining resource reasoning and mutual exclusion

- Uses critical path and mutual exclusion analysis to propagate integrated resource bounds

Bounding resource usage

Using maximal flow to calculate tight bounds

- Given a temporal network of resource use events, determine max/min resource use at a given time T
- Identify events that can be ordered with respect to time T
- Build flow network from events and resource use
- Maximal flow calculations provide resource bound
- Bounds are provably tight

Ongoing work

- Theoretical results and algorithms in place
- Incorporation into planning framework and performance tests to be done in near future

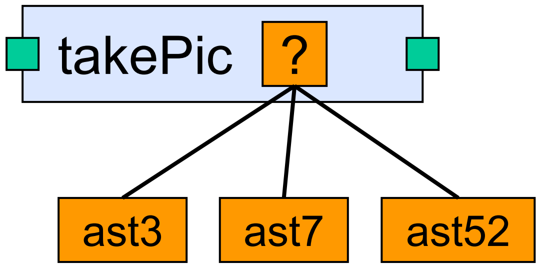
Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ✓ Simple Temporal Network Reasoning
- ✓ Procedural Constraint Reasoning
- ✓ Resource Reasoning
- ❑ Mapping Planning into Dynamic Constraint Satisfaction
  - ❑ Objects, Tokens, Constraints
  - ❑ Subgoaling
- ❑ Example

Constraint-Based Planning

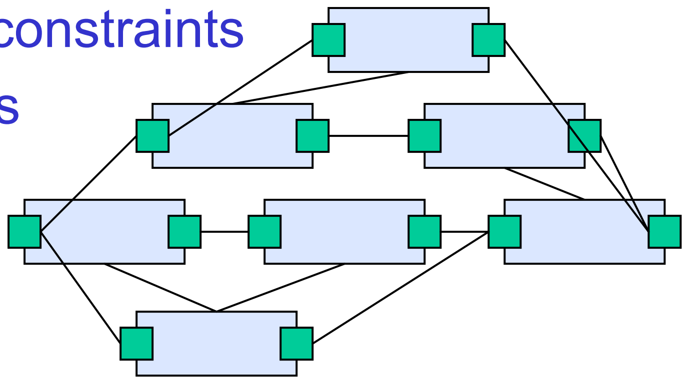
Activities represented as intervals

- Each interval specifies activity
- Each interval has start and end
- Interval can have parameters
- Parameter variables have domains



Candidate plan is a network of intervals

- Start and end times linked by temporal constraints
- Interval parameters linked by constraints
- Gives rise to constraint network

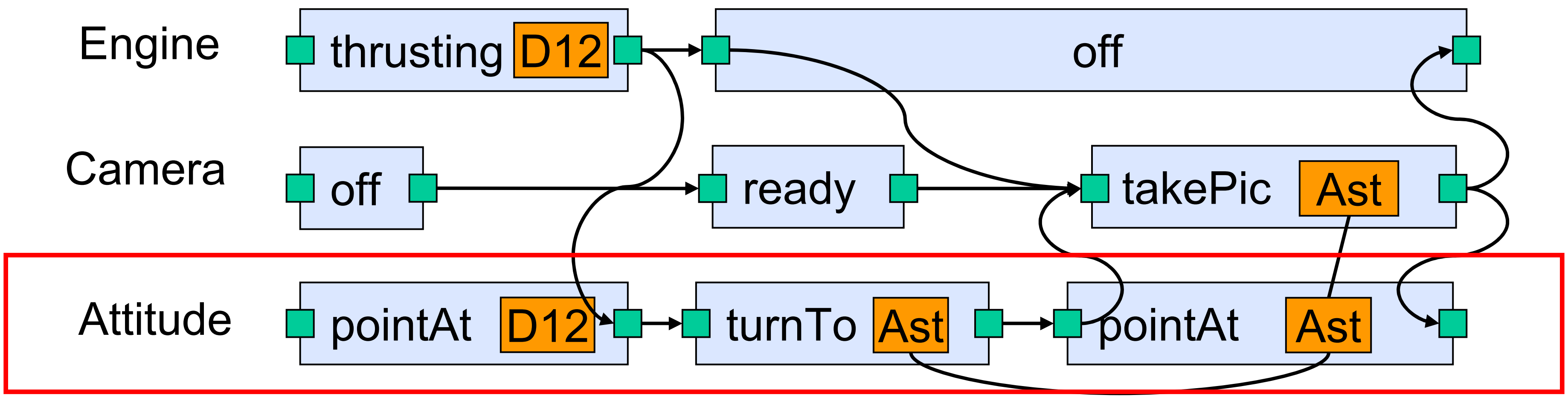


Feasibility of candidate plan

- If network is inconsistent, cannot become a valid plan

Plan Representation

A network of Tokens, linked by constraints



Flexible Time Intervals have Flexible Start, End and Duration

Parameterized Predicates describe actions and states

Token is a Parameterized Predicate over Flexible Time Interval

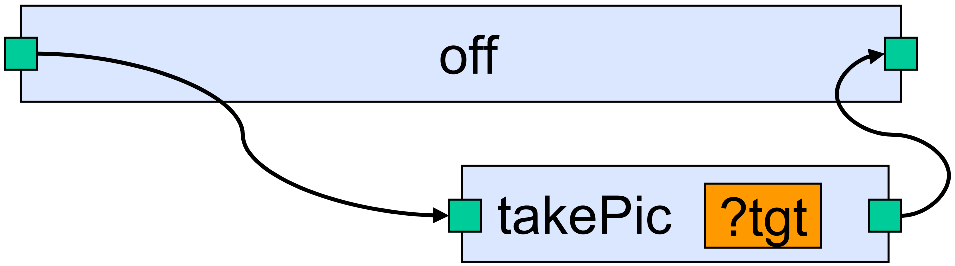
Constraints defined between Tokens, Time Points, Parameters

Objects enforce constraints over Tokens

Temporal Constraints

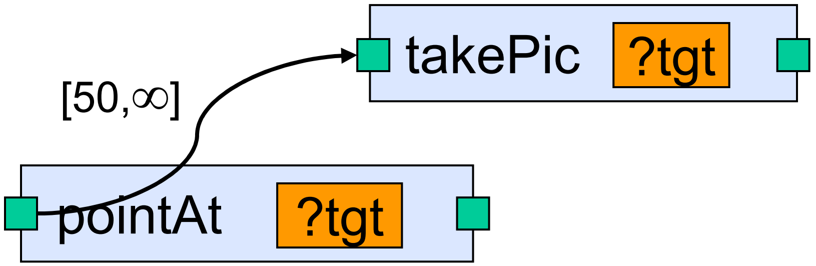
Qualitative relations (Allen)

- before,after,
  contains,contained by,
- ....
- Example:
  takePic contained by off



Quantitative bounds

- Example:
  pointAt starts at least 50 seconds before takePic

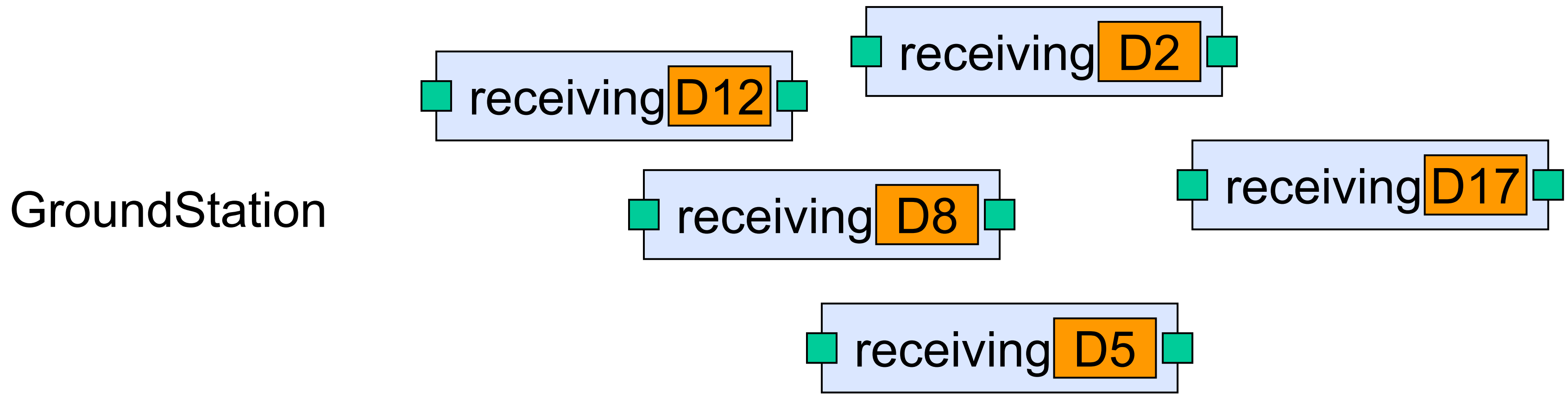


Objects

Objects impose no ordering constraints on its predicates.

Useful to keep track of events that occur in time with no restrictions on when they occur or how many of them occur at any one time.

Example:



Static Objects

Objects with properties but no predicates.

Useful to represent entities in the system that don’t change with respect to time.

Example: Targets

Target

X_pos

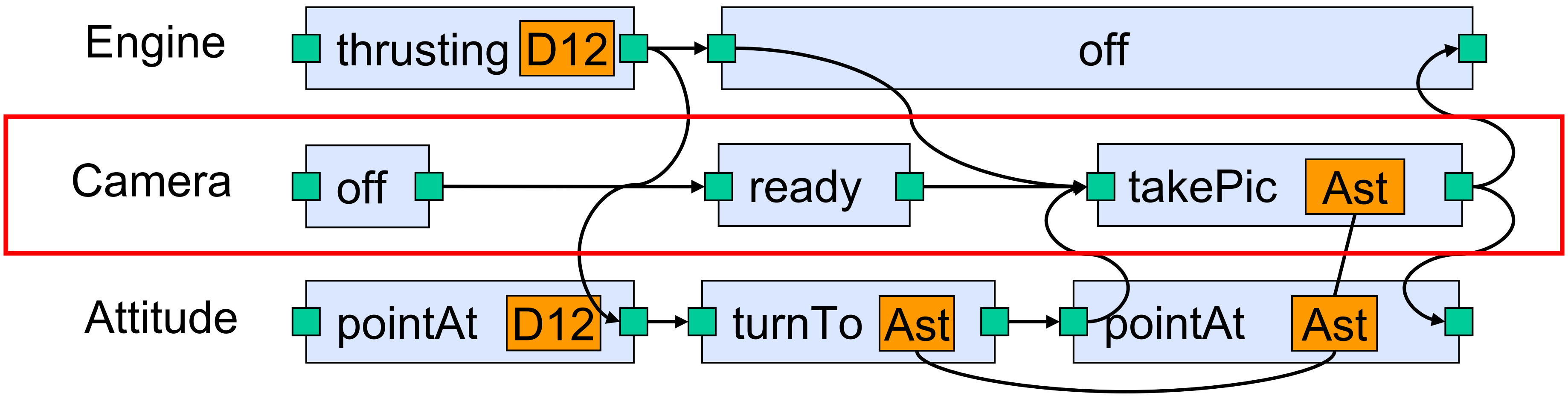
Y_pos

Z_pos

Timelines

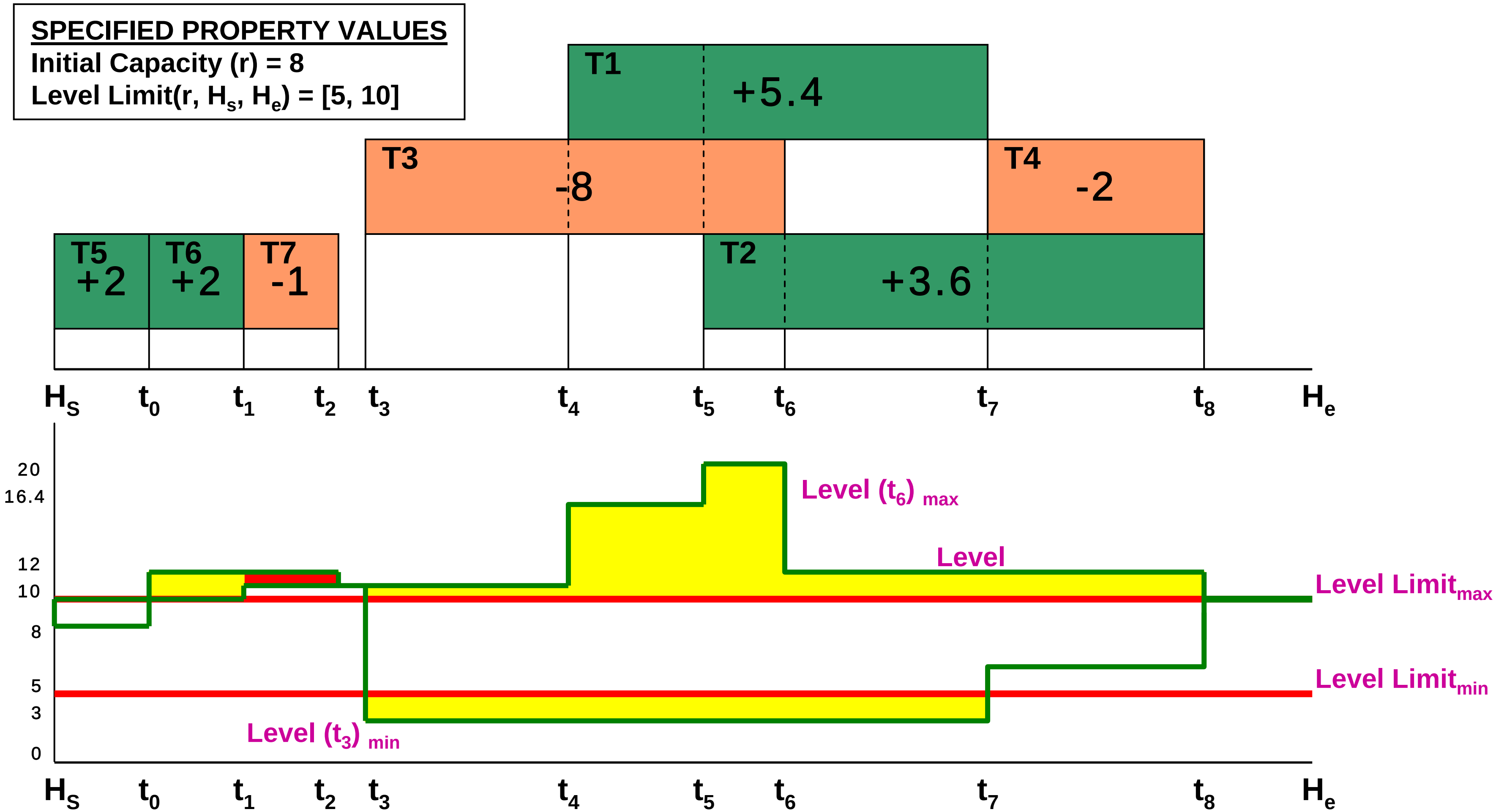
Enforce that activities for same system do not conflict

- Activities on same timelines are temporally ordered
- Mutual exclusion constraints apply between activities



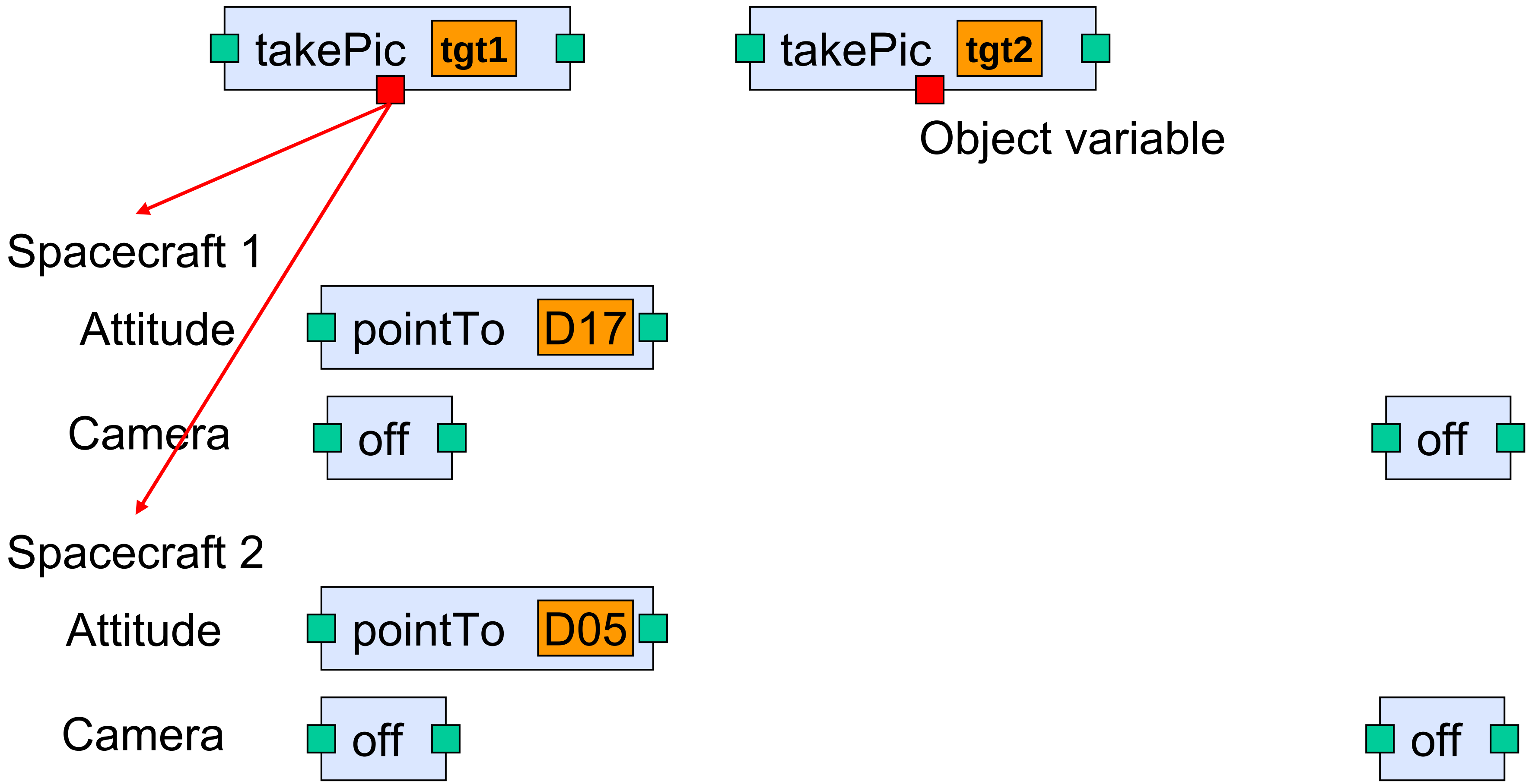
Resources

Enforce capacity, production and consumption limits (predicates are instantaneous with flexible start time)



Multiple objects of same class

Tokens have the following variables:
object, state, start, end, duration plus any parameters



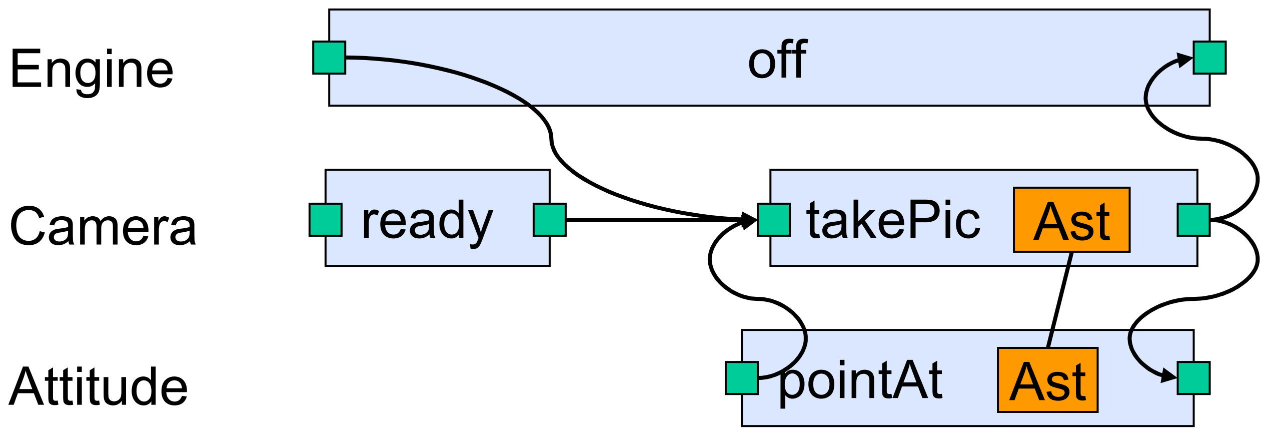
Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ✓ Simple Temporal Network Reasoning
- ✓ Procedural Constraint Reasoning
- ✓ Resource Reasoning
- ❑ Mapping Planning into Dynamic Constraint Satisfaction
  - ✓ Objects, Tokens, Constraints
  - ❑ Subgoaling
- ❑ Example

Subgoaling Constraints

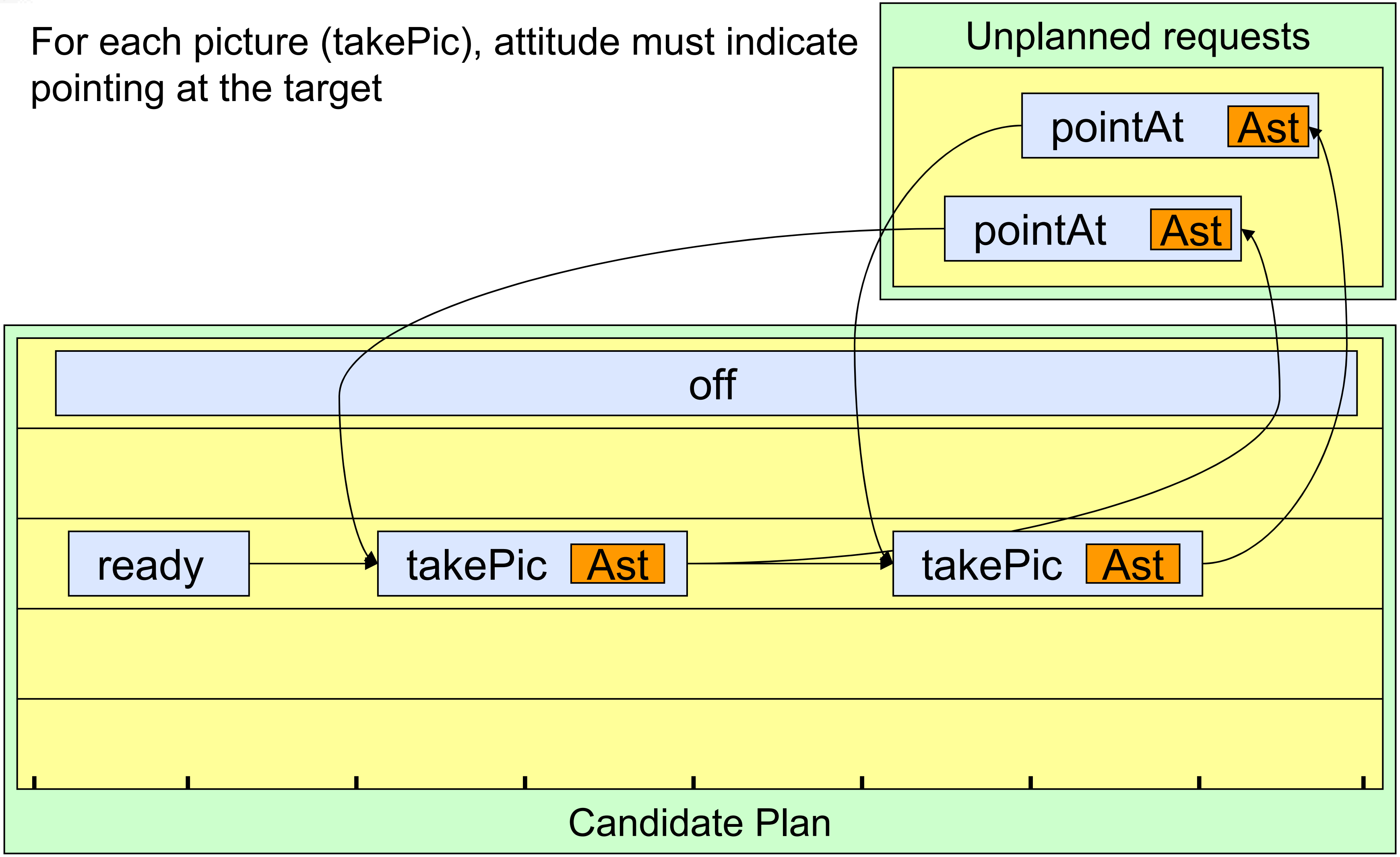
Subgoals

- Other activities may be required to be in the plan to successfully perform an activity
- Example: To take a picture you'll need:



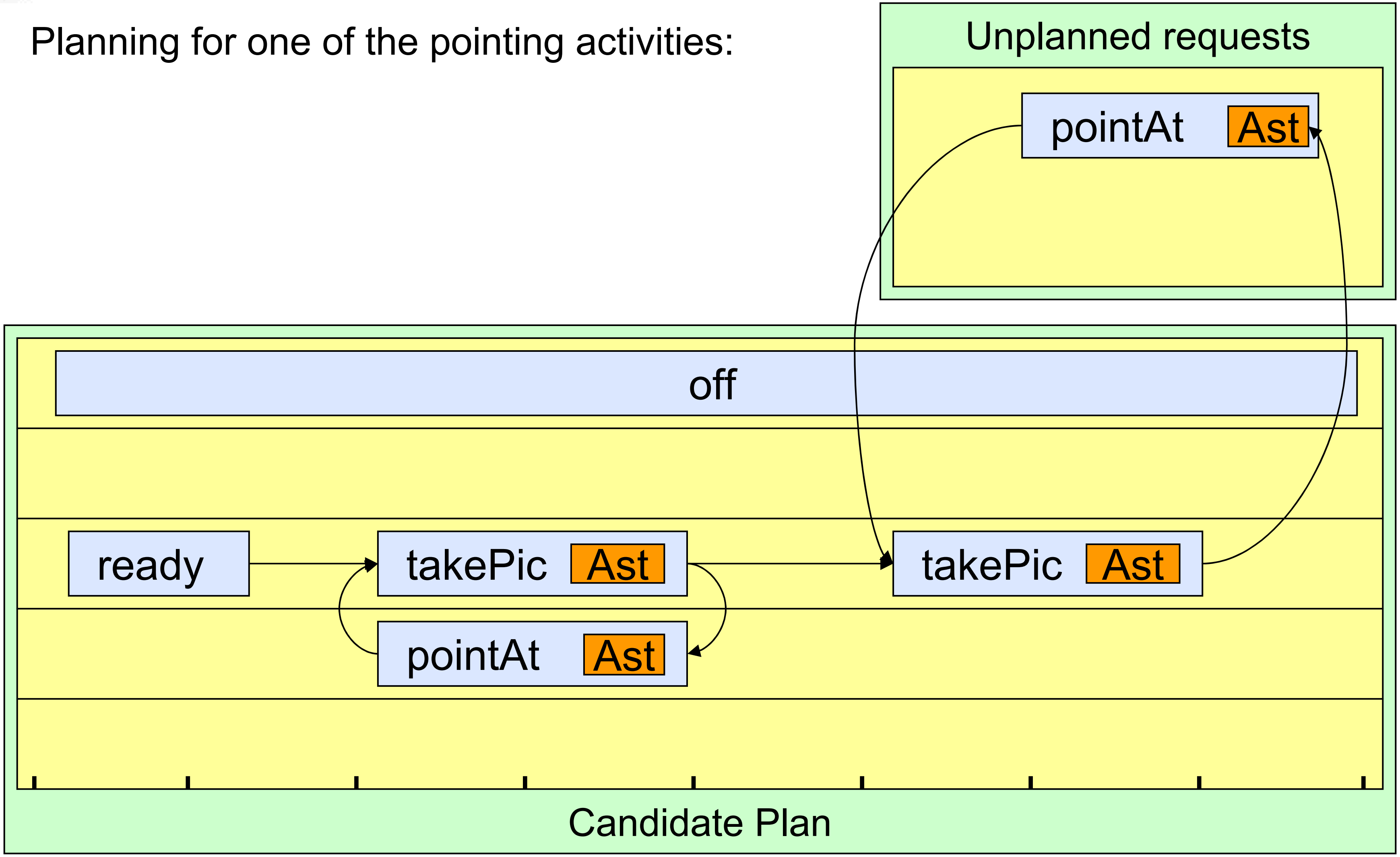
Support activities needed

For each picture (takePic), attitude must indicate pointing at the target



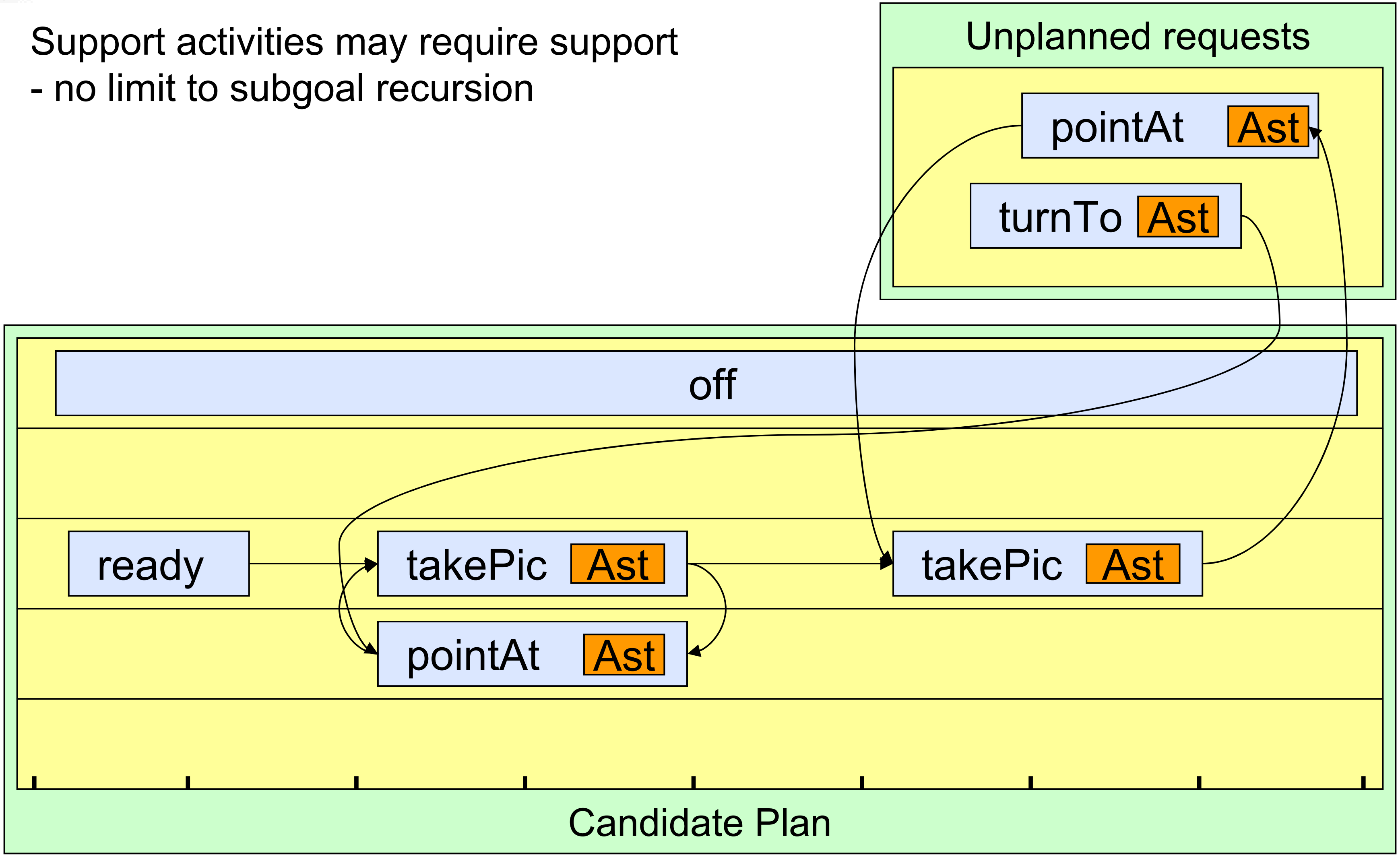
Planning Support Activities

Planning for one of the pointing activities:



Recursive Support Activities

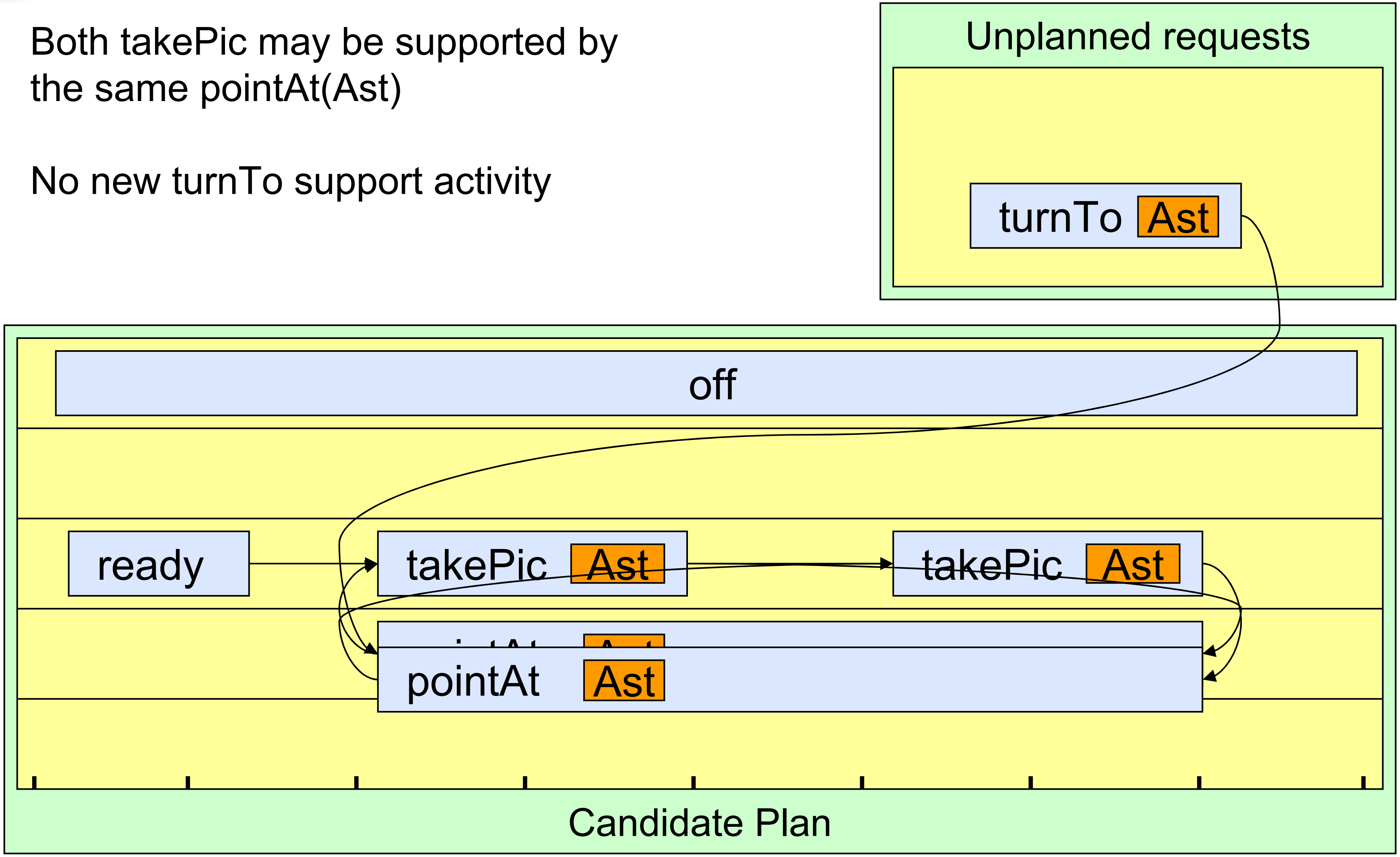
Support activities may require support
- no limit to subgoal recursion



Only necessary support added

Both takePic may be supported by the same pointAt(Ast)

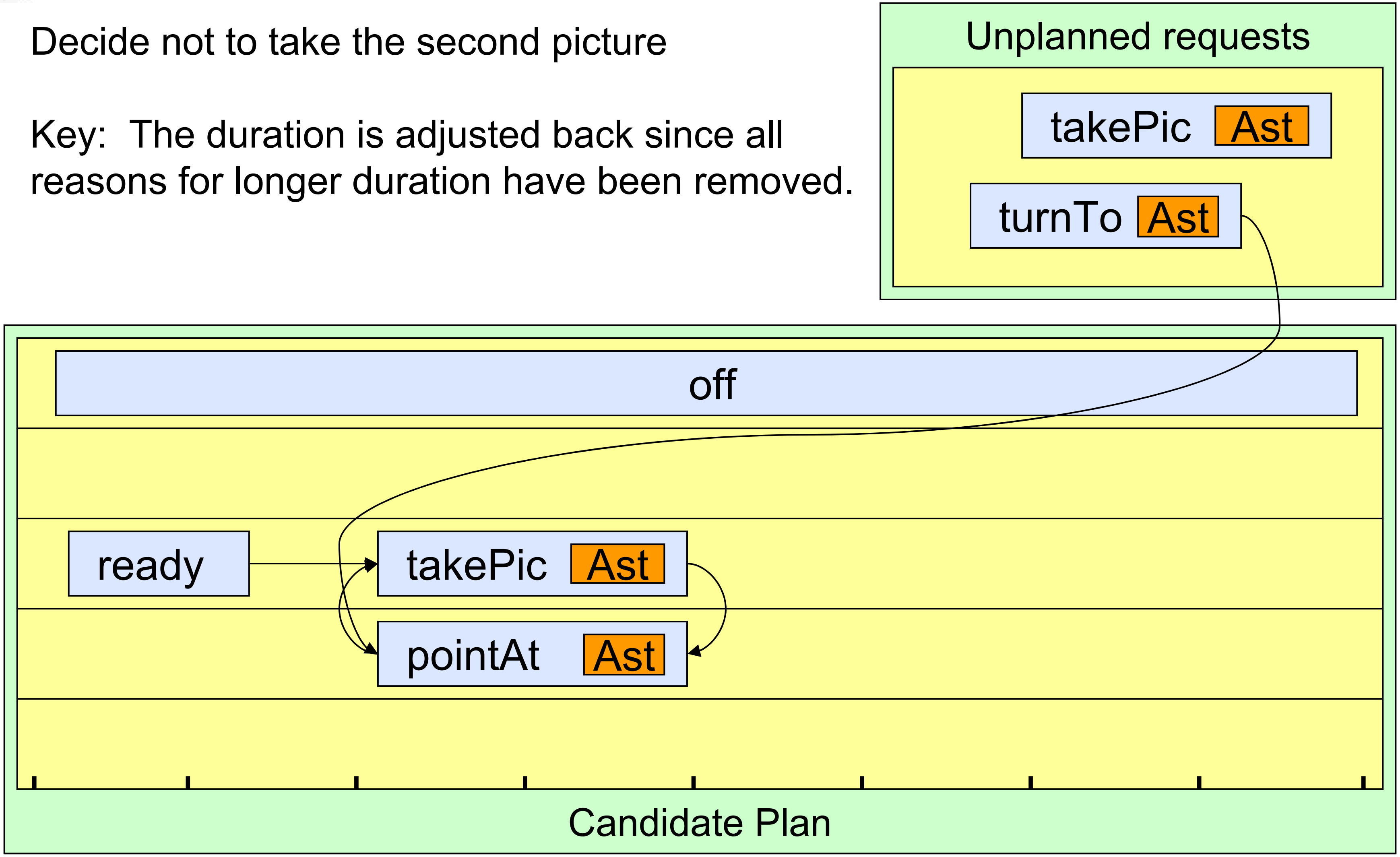
No new turnTo support activity



Support requests updated

Decide not to take the second picture

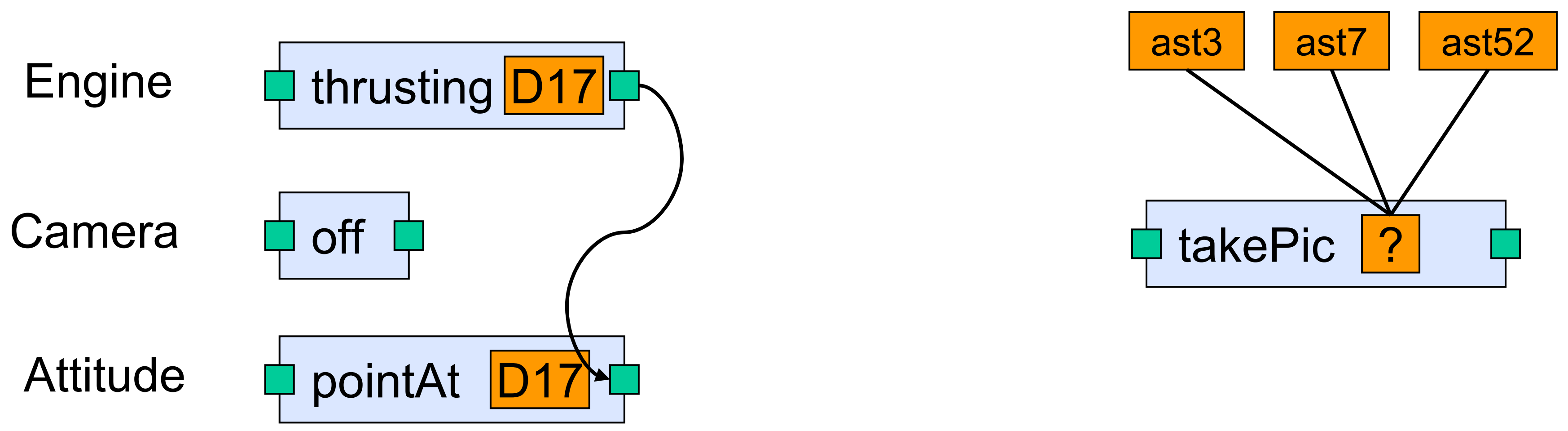
Key: The duration is adjusted back since all reasons for longer duration have been removed.



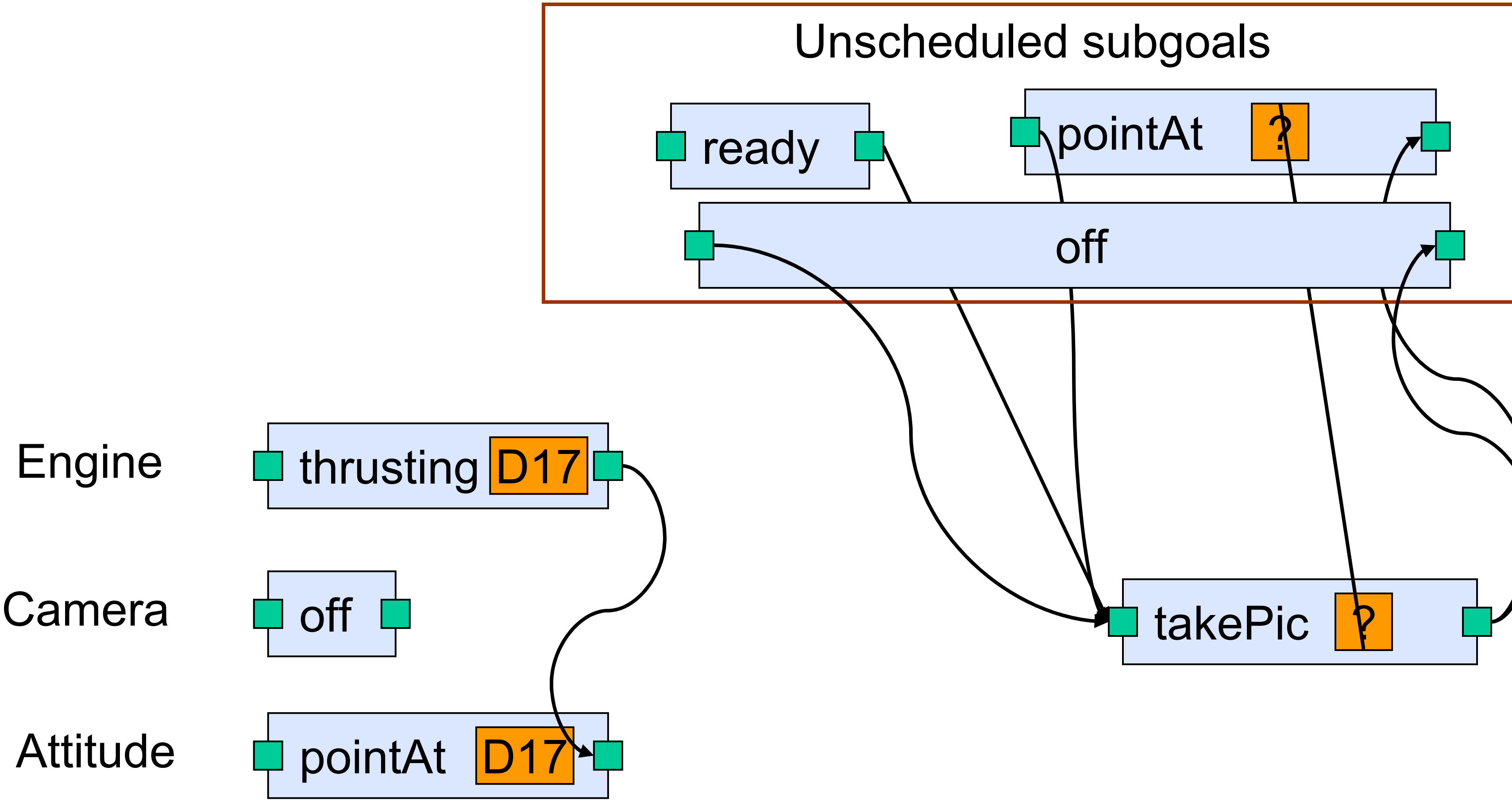
Background on Constraint Based Planning

- ✓ Constraint Satisfaction Reasoning
- ✓ Simple Temporal Network Reasoning
- ✓ Procedural Constraint Reasoning
- ✓ Resource Reasoning
- ✓ Mapping Planning into Dynamic Constraint Satisfaction
  - ✓ Objects, Tokens, Constraints
  - ✓ Subgoaling
- ❑ Example

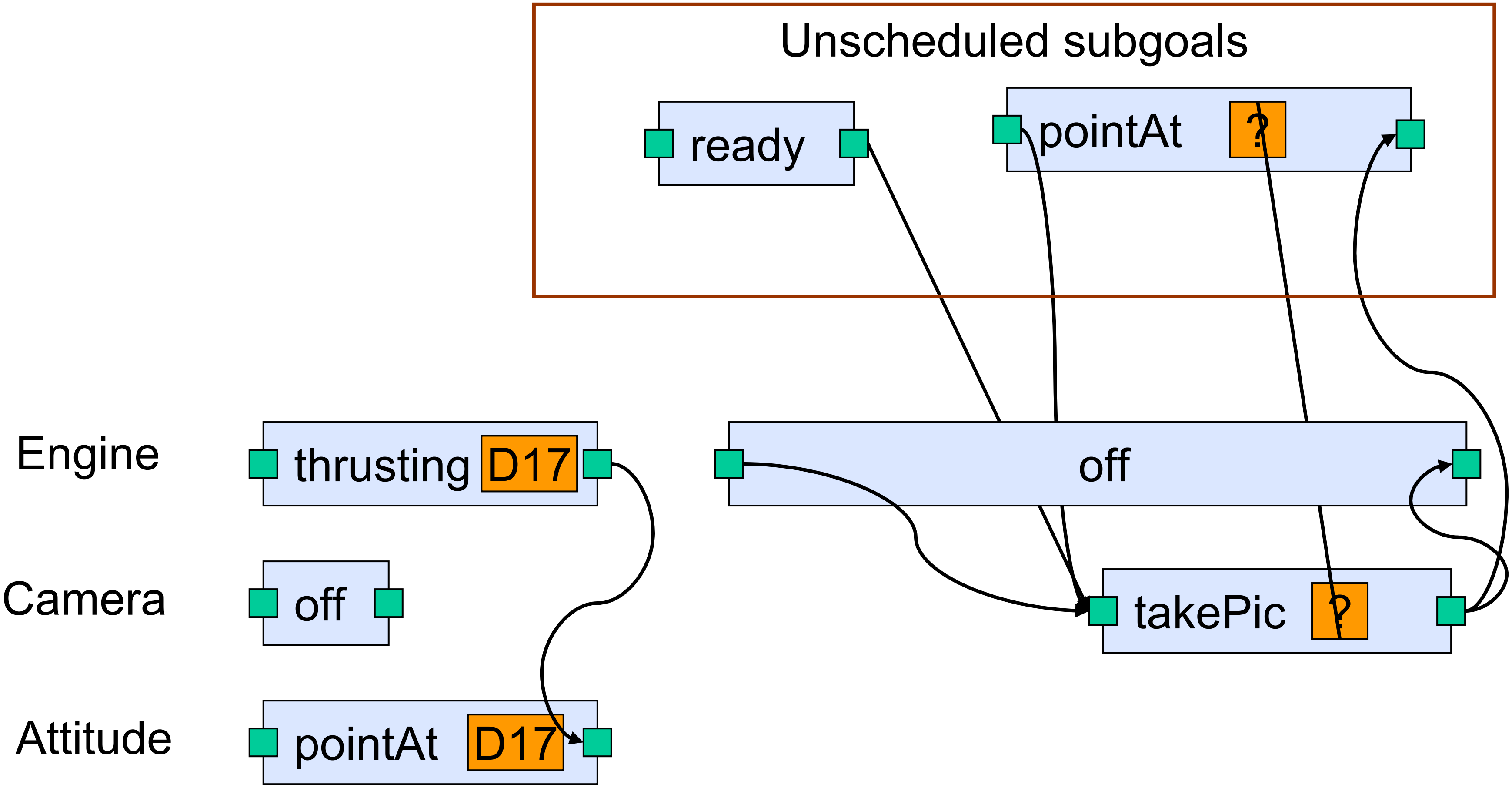
Initial State



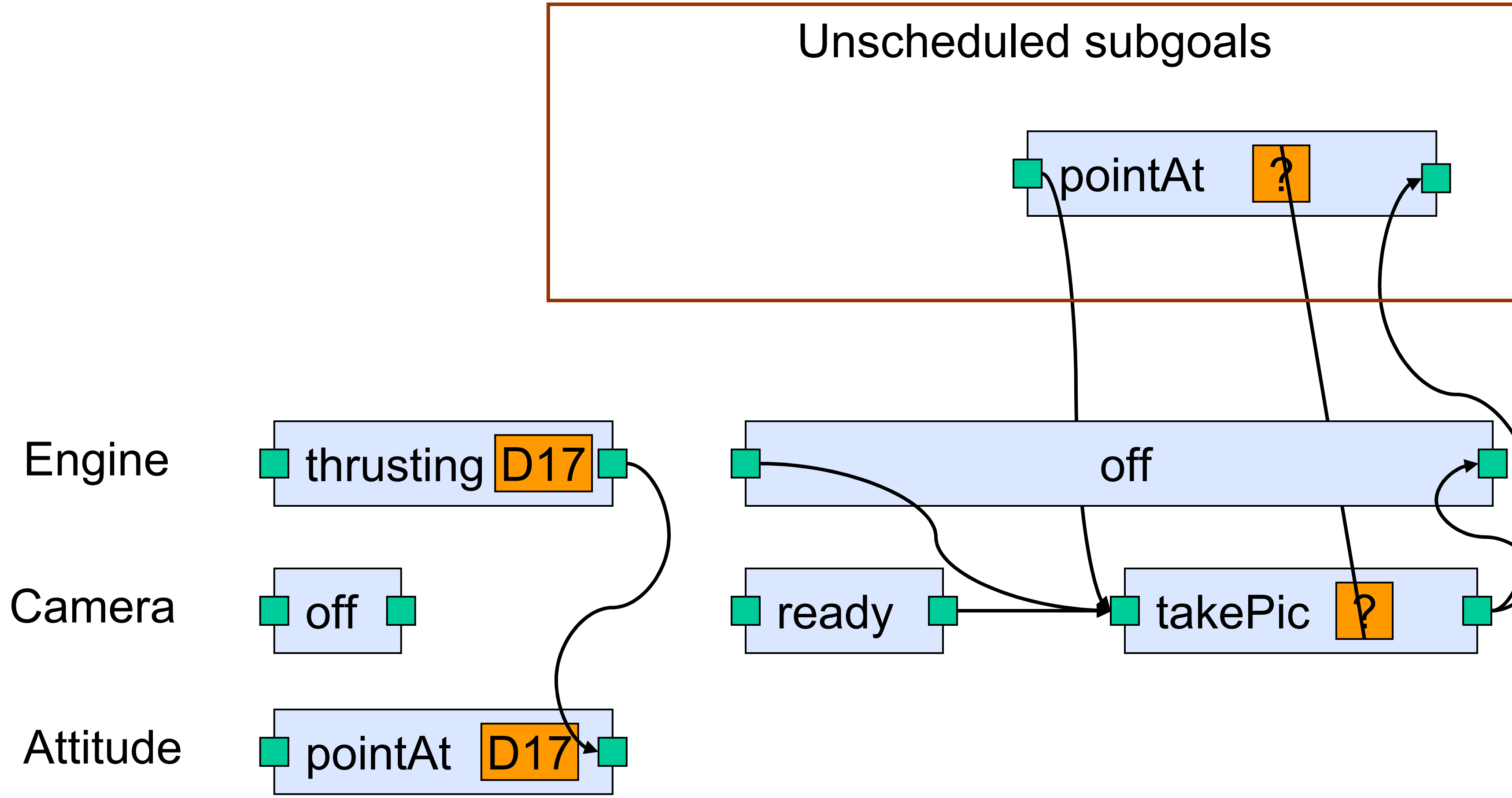
Expand takePic Subgoals



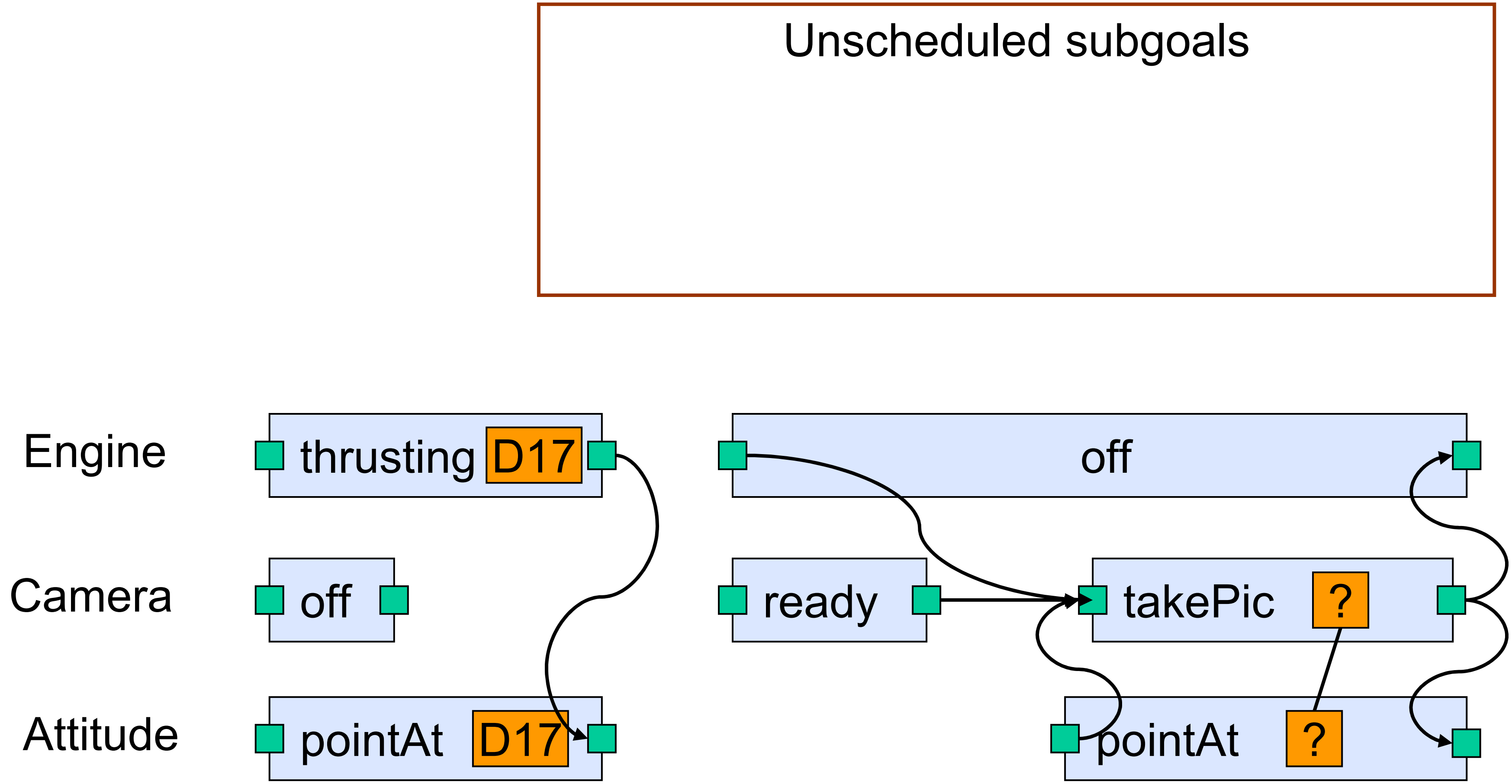
Insert off subgoal



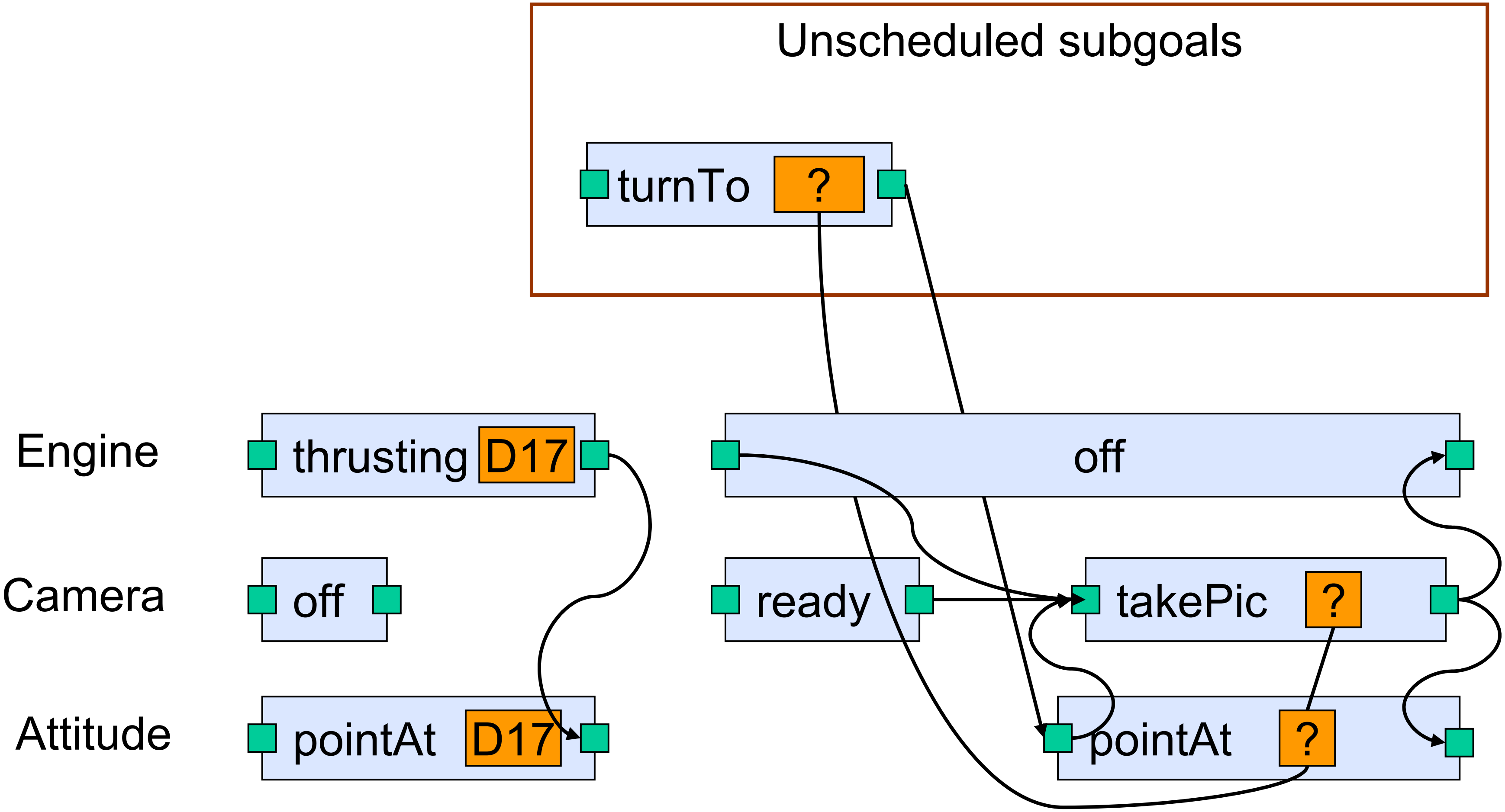
Insert ready Subgoal



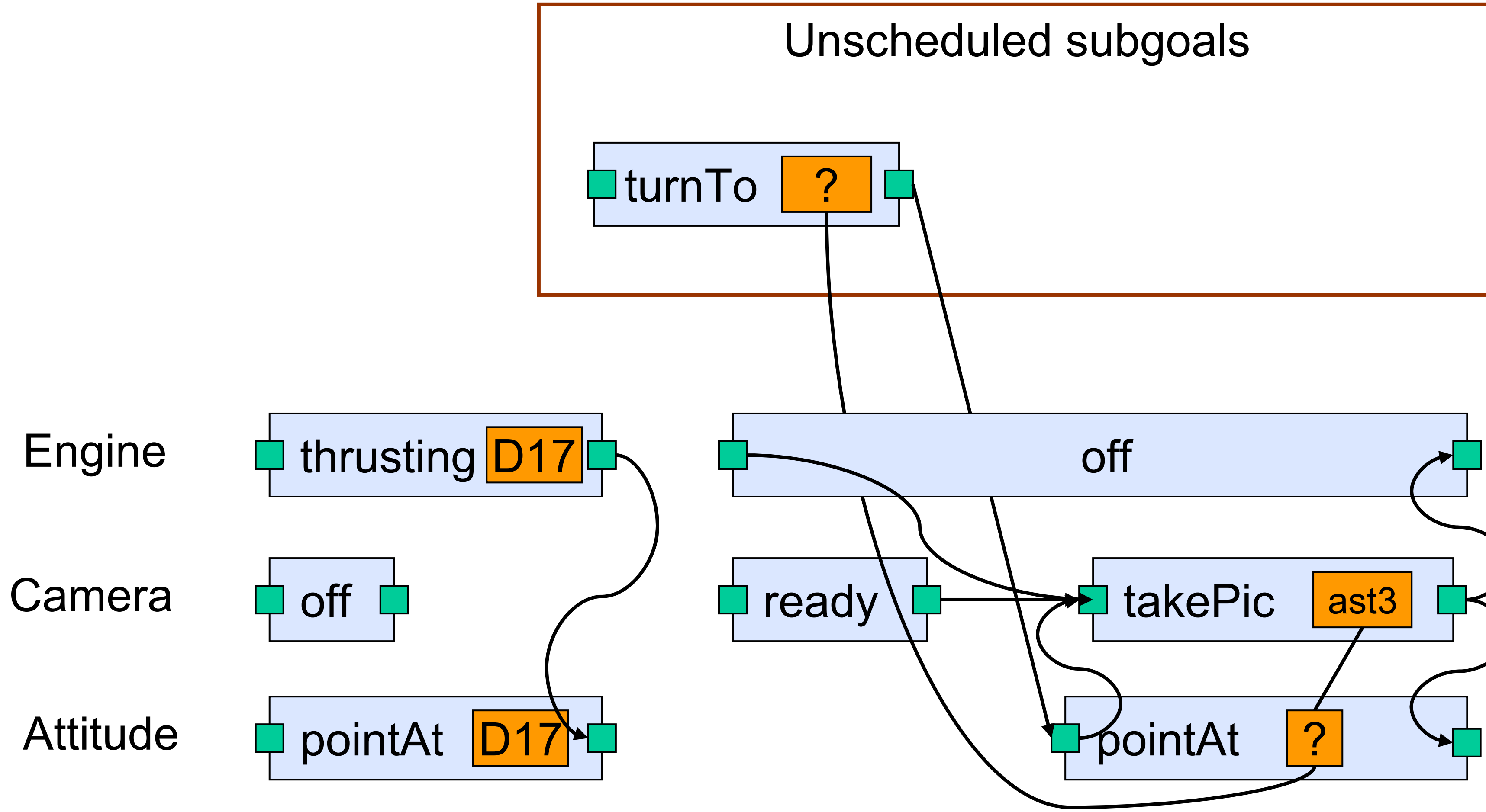
Insert pointAt Interval



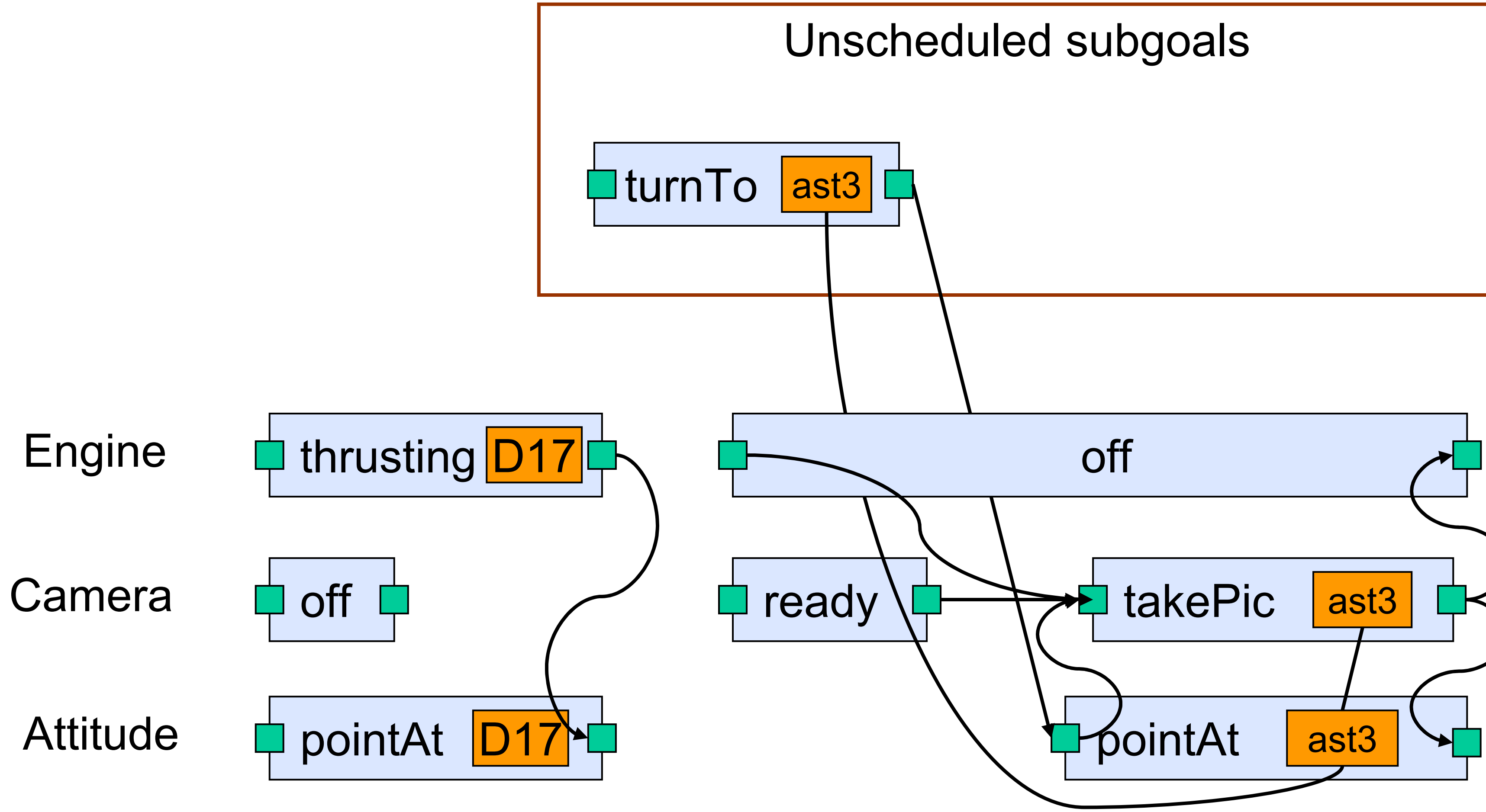
Expand pointAt



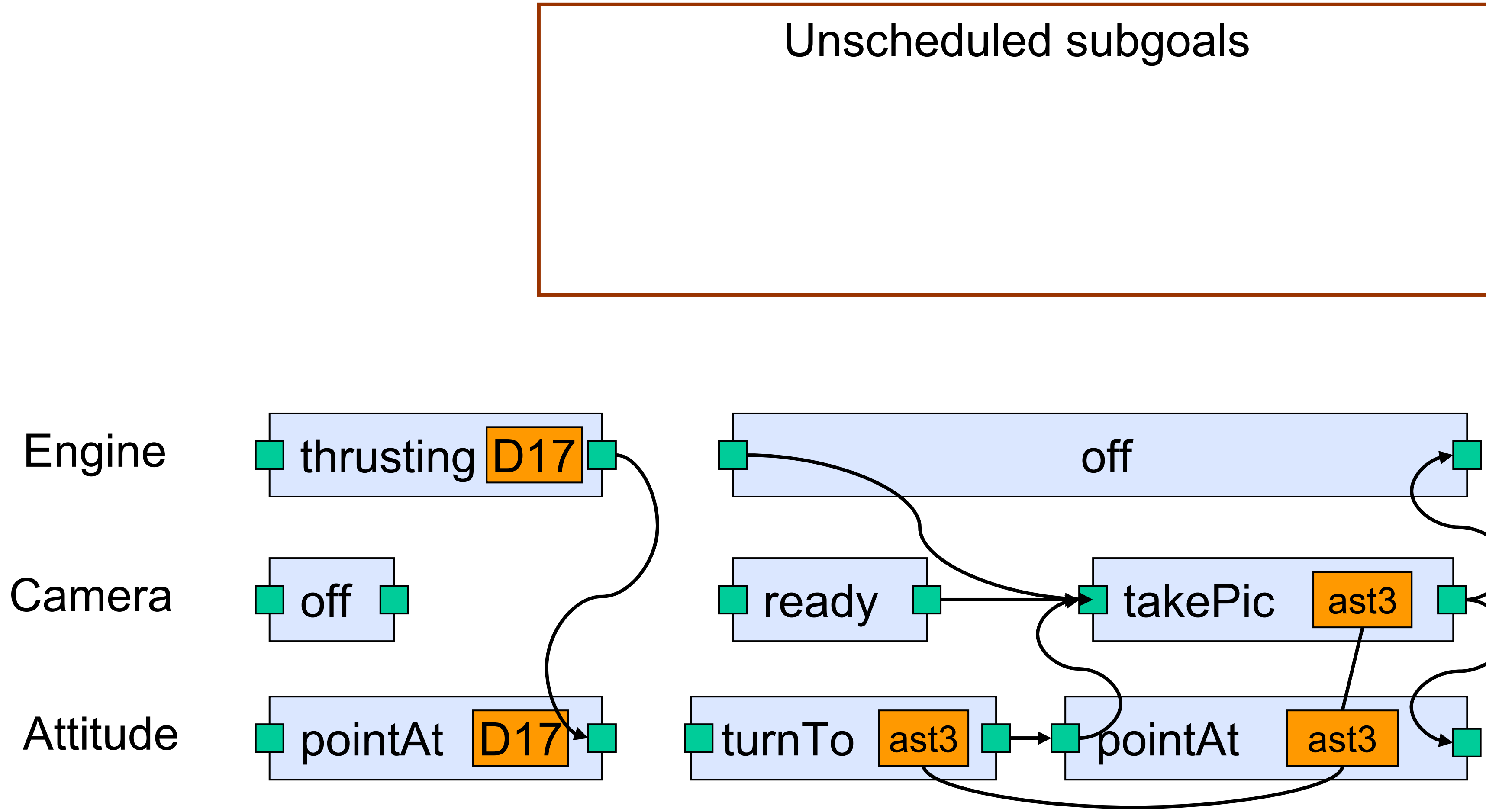
Select image Target



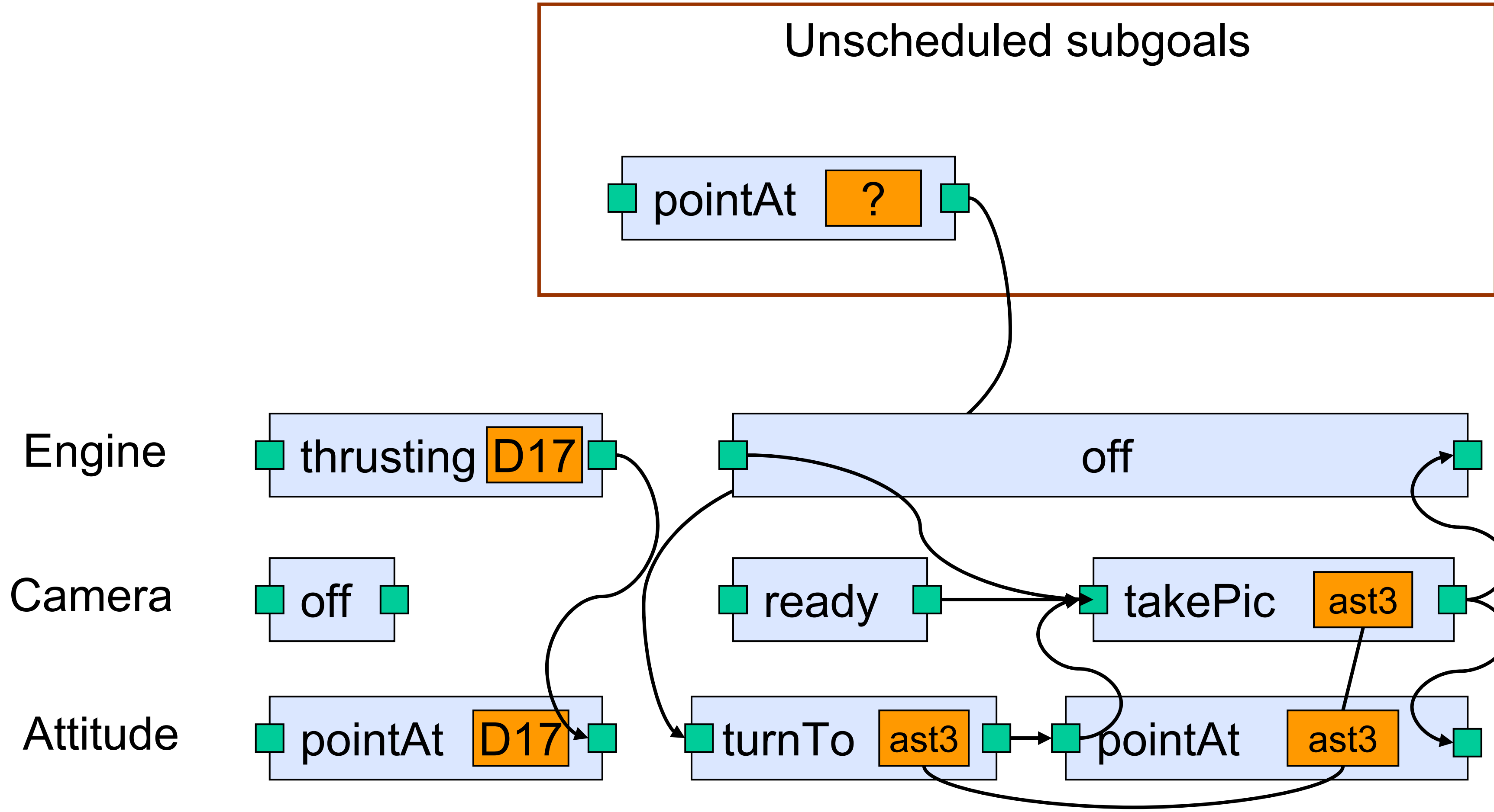
Propagate Consequences



Insert turnTo

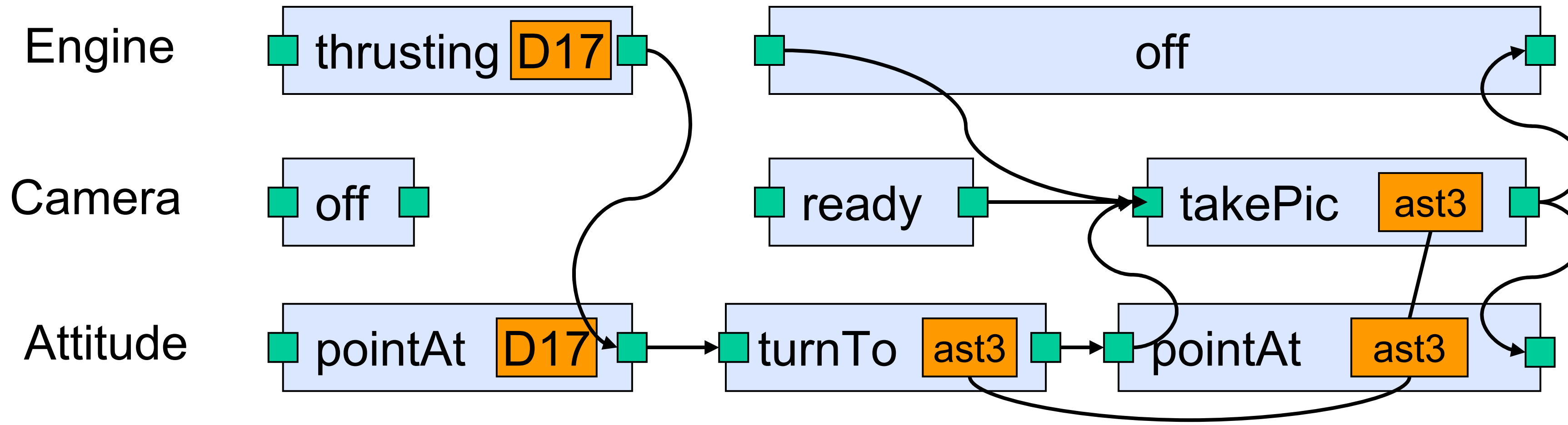


Expand turnTo

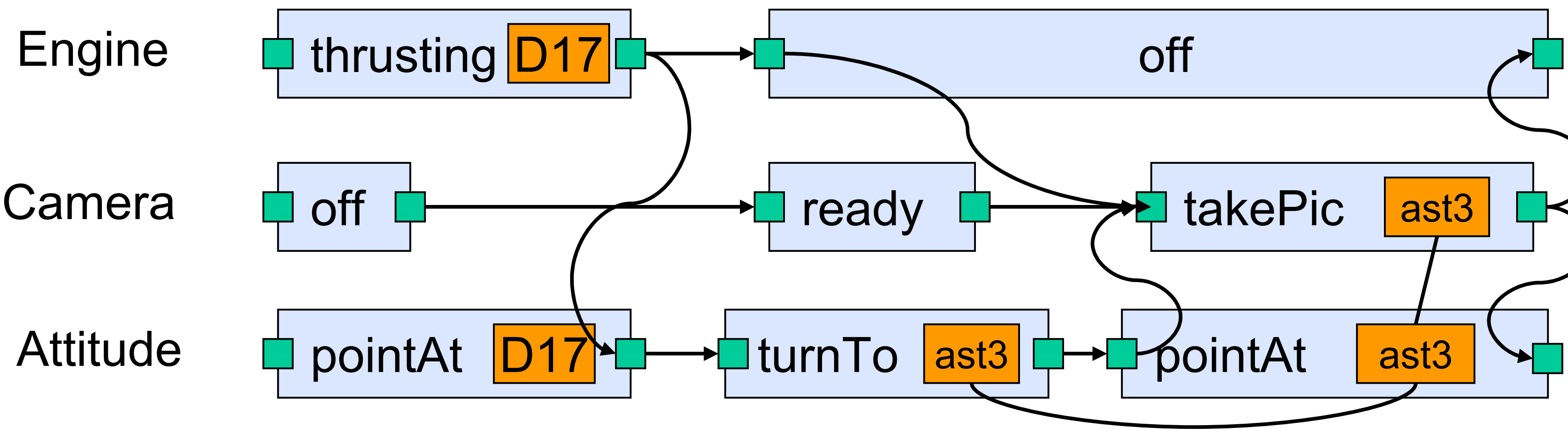


Coalesce pointAt Goals

Unscheduled subgoals



Final Plan



Engine
thrusting
D17
off
Camera
off
ready
takePic
ast3
Attitude
pointAt
D17
turnTo
ast3
pointAt
ast3

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ❑ Architecture
- ❑ NDDL – New Domain Description Language
- ❑ Assemblies
- ❑ PlanWorks
- ❑ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Europa 2

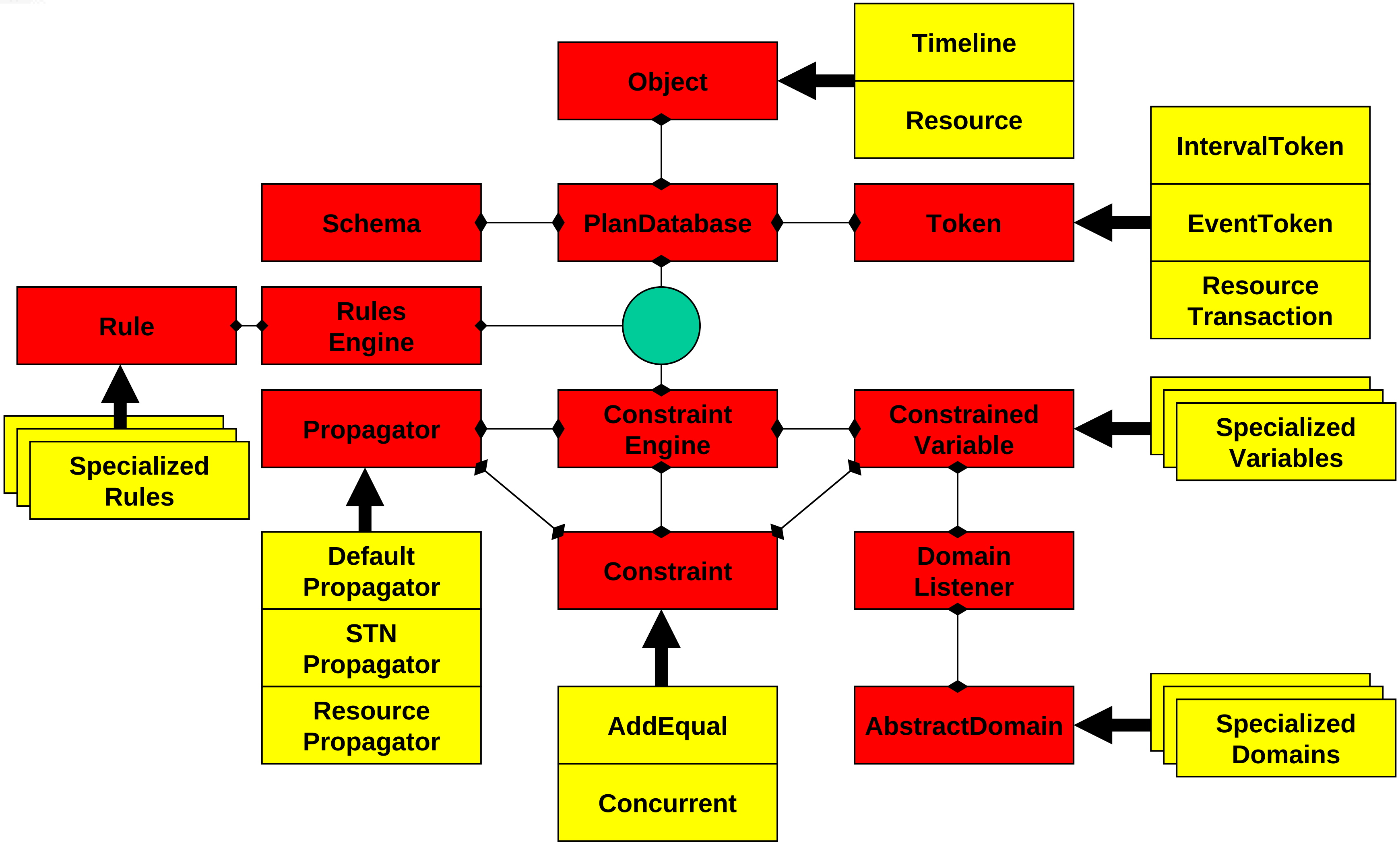
Modules:

- Plan Database
- Constraint Engine
- Temporal Network
- Resources
- Rules Engine
- Chronological Backtracking Planner
- NDDL
- Utils
- Aver.

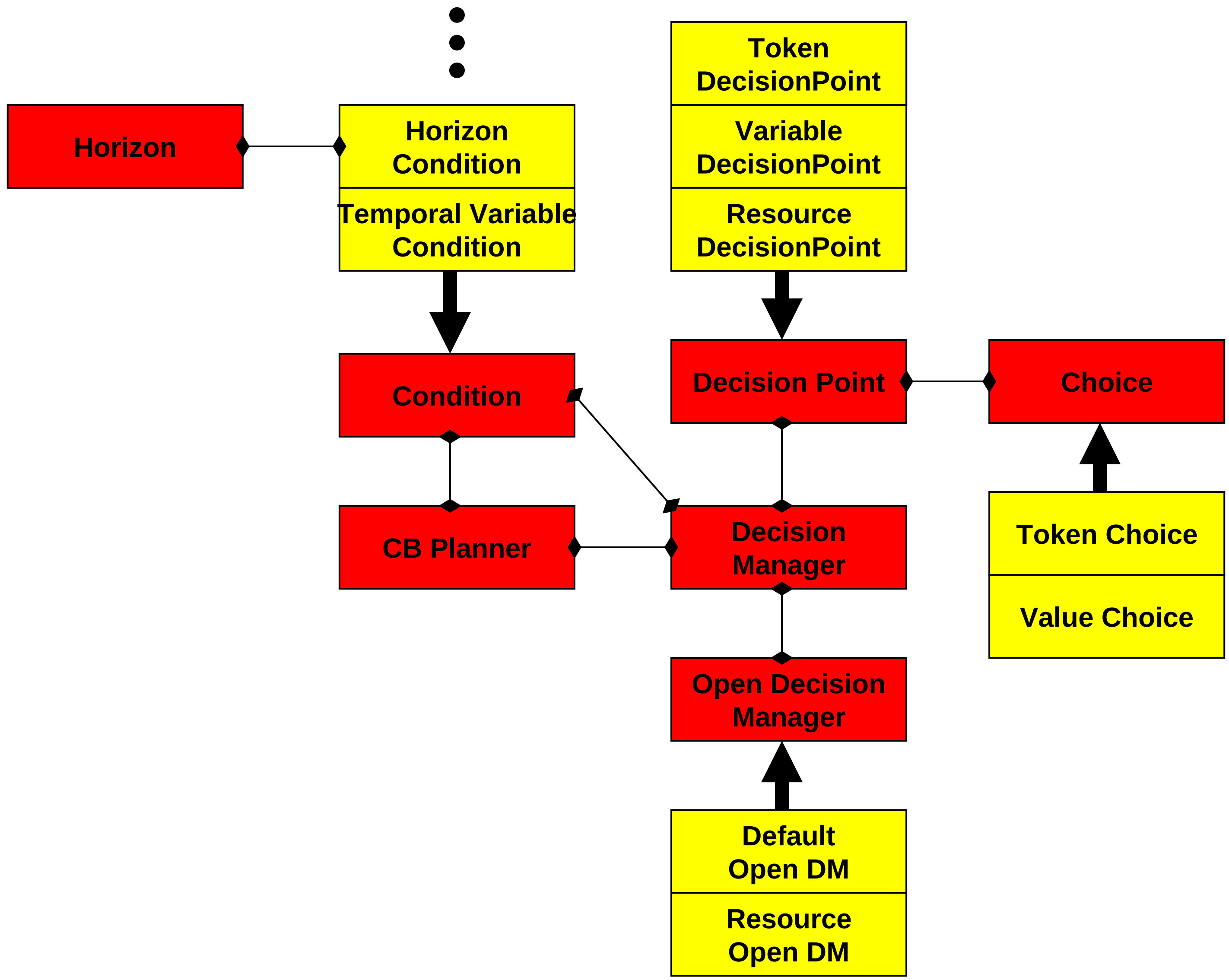
Architecture Principles:

- General framework for **extensibility**
- Highly **efficient** implementations
- Behaviors can be turned on and off for **flexibility**
- Dependencies managed through event listeners and adapters

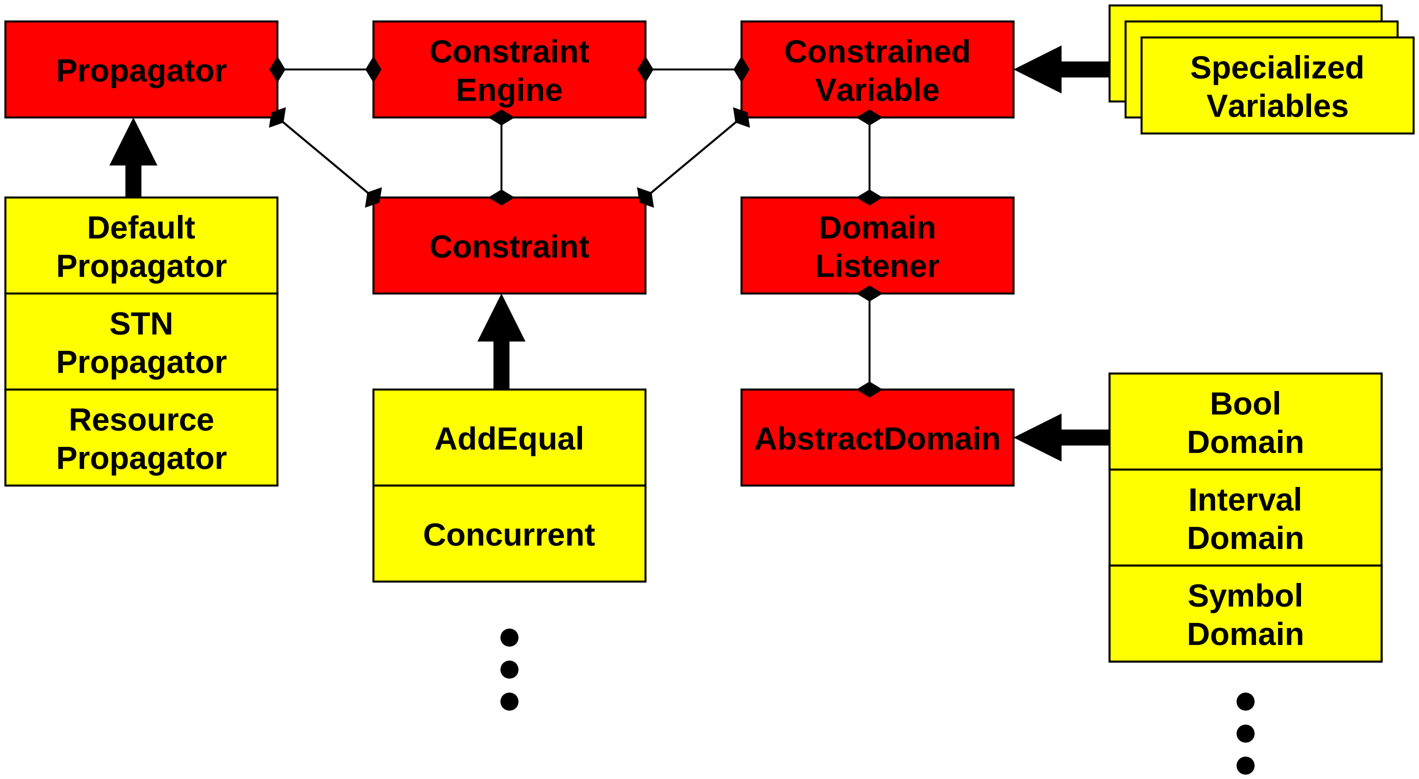
Europa 2
Framework & Components



Europa 2
Chronological Backtracking Planner



Constraint Engine



Domain Representation

AbstractDomain base class provides interface

Domain specializations:

- EnumeratedDomain
  - NumericDomain
  - StringDomain
  - SymbolDomain
- IntervalDomain
  - IntervalIntDomain
    - BoolDomain

Strong type checking is enforced throughout.

Type factories should be used to create variables, domains, and values of specific types.

Decision Model

Variable Decisions (resolve unbound variables)

- specify(var,val) / reset(var)

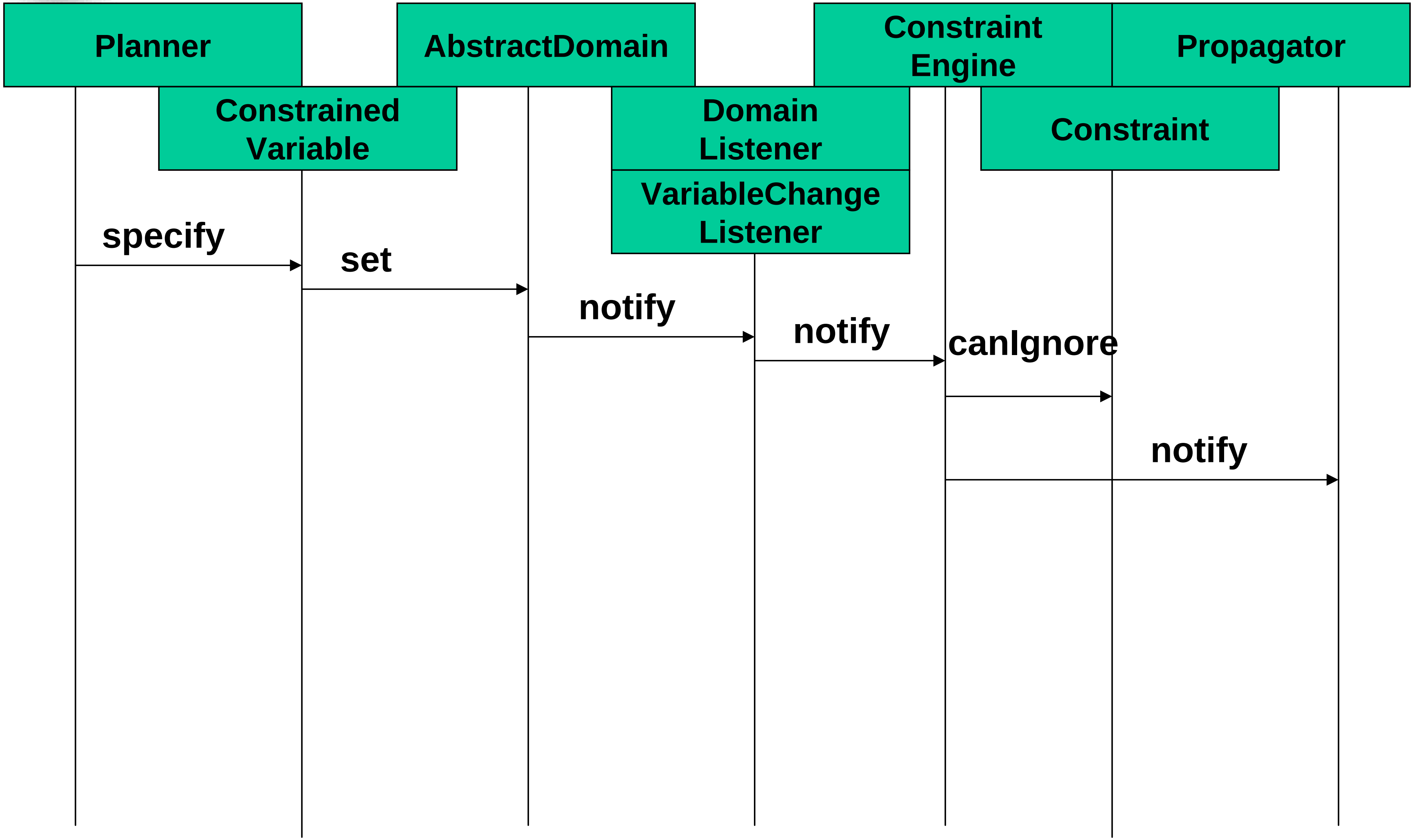
Token Decisions (resolve inactive tokens)

- activate(token) / deactivate(token)
- merge(inactiveToken,activeToken) / split(inactiveToken)
- reject(token) / reinstate(token)

Object Decisions (resolve object with unordered tokens)

- constrain(object,predecessorToken,successorToken)
- free(object,predecessorToken,successorToken)

Constraint Engine Framework
Specify: Collaboration Diagram



Domain Change Events

Notifications:

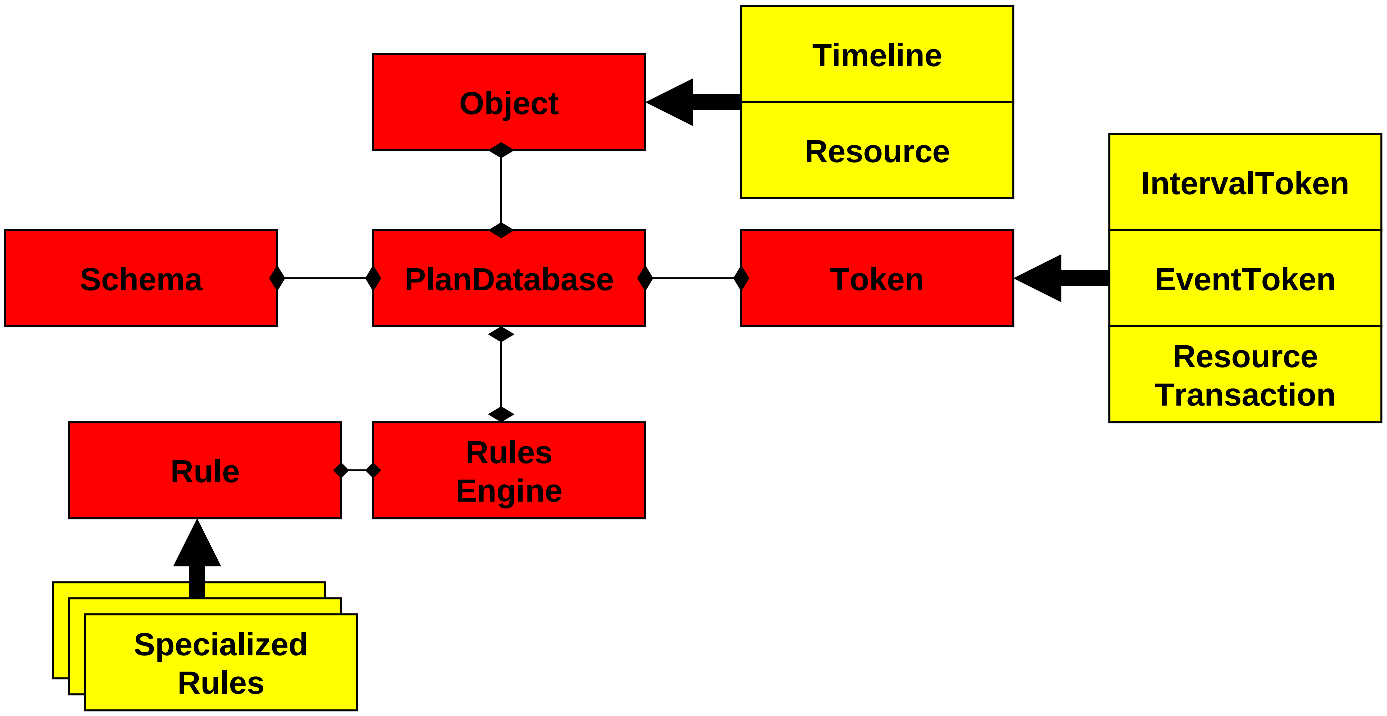
- UPPER_BOUND_DECREASED
- LOWER_BOUND_INCREASED
- VALUE_REMOVED
- RESTRICT_TO_SINGLETON
- SET
- SET_TO_SINGLETON
- RESET
- RELAXED
- CLOSED
- EMPTIED

Constraint Engine Events

Notifications:

- PROPAGATION_COMMENCED
- PROPAGATION_COMPLETED
- PROPAGATION_PREEMPTED
- CONSTRAINT_ACTIVATED / DEACTIVATED
- VARIABLE_ACTIVATED / DEACTIVATED
- VARIABLE_ADDED / REMOVED
- CONSTRAINT_ADDED / REMOVED
- CONSTRAINT_EXECUTED

Plan Database



Decision Model

Variable Decisions (resolve unbound variables)

- specify(var,val) / reset(var)

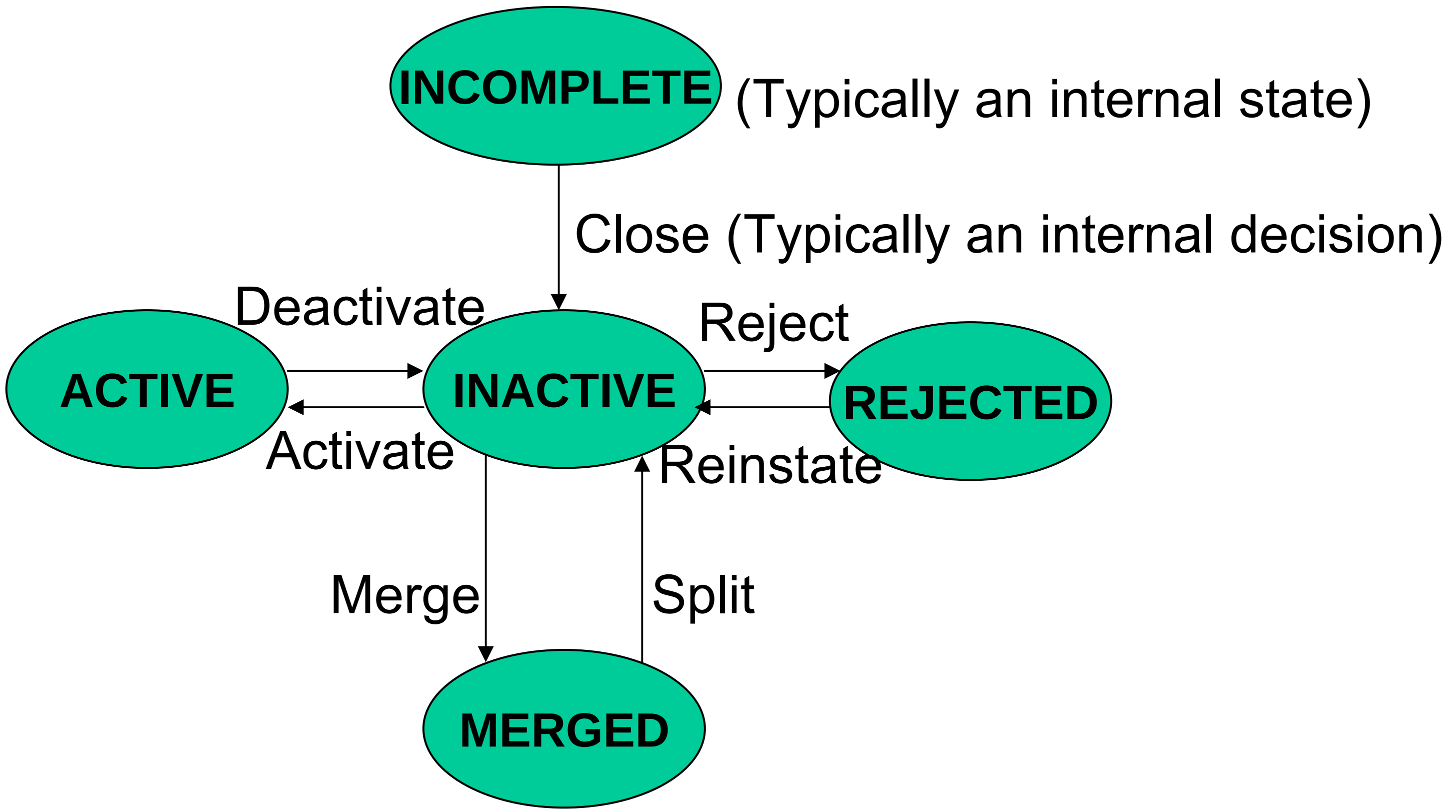
Token Decisions (resolve inactive tokens)

- activate(token) / deactivate(token)
- merge(inactiveToken,activeToken) / split(inactiveToken)
- reject(token) / reinstate(token)

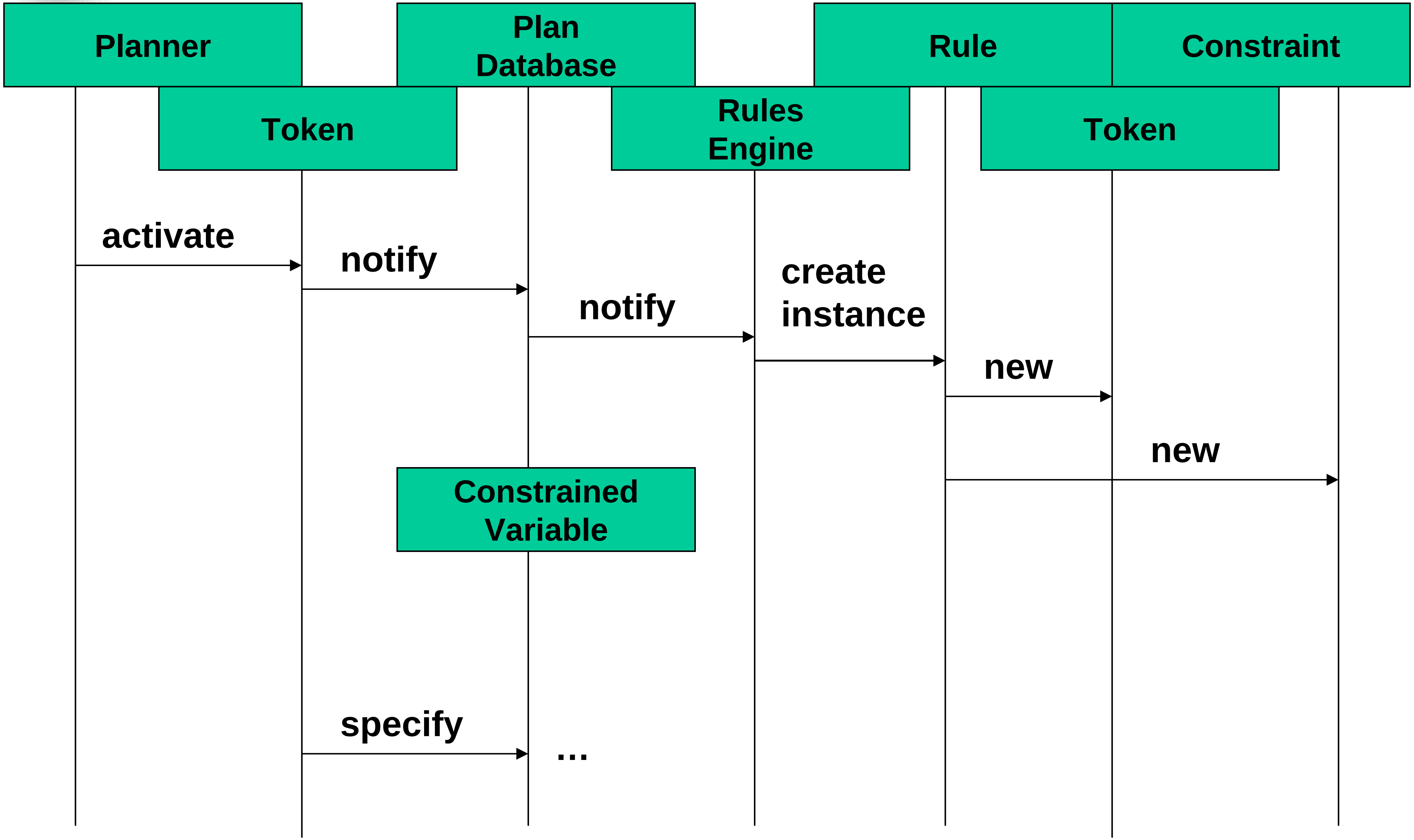
Object Decisions (resolve object with unordered tokens)

- constrain(object,predecessorToken,successorToken)
- free(object,predecessorToken,successorToken)

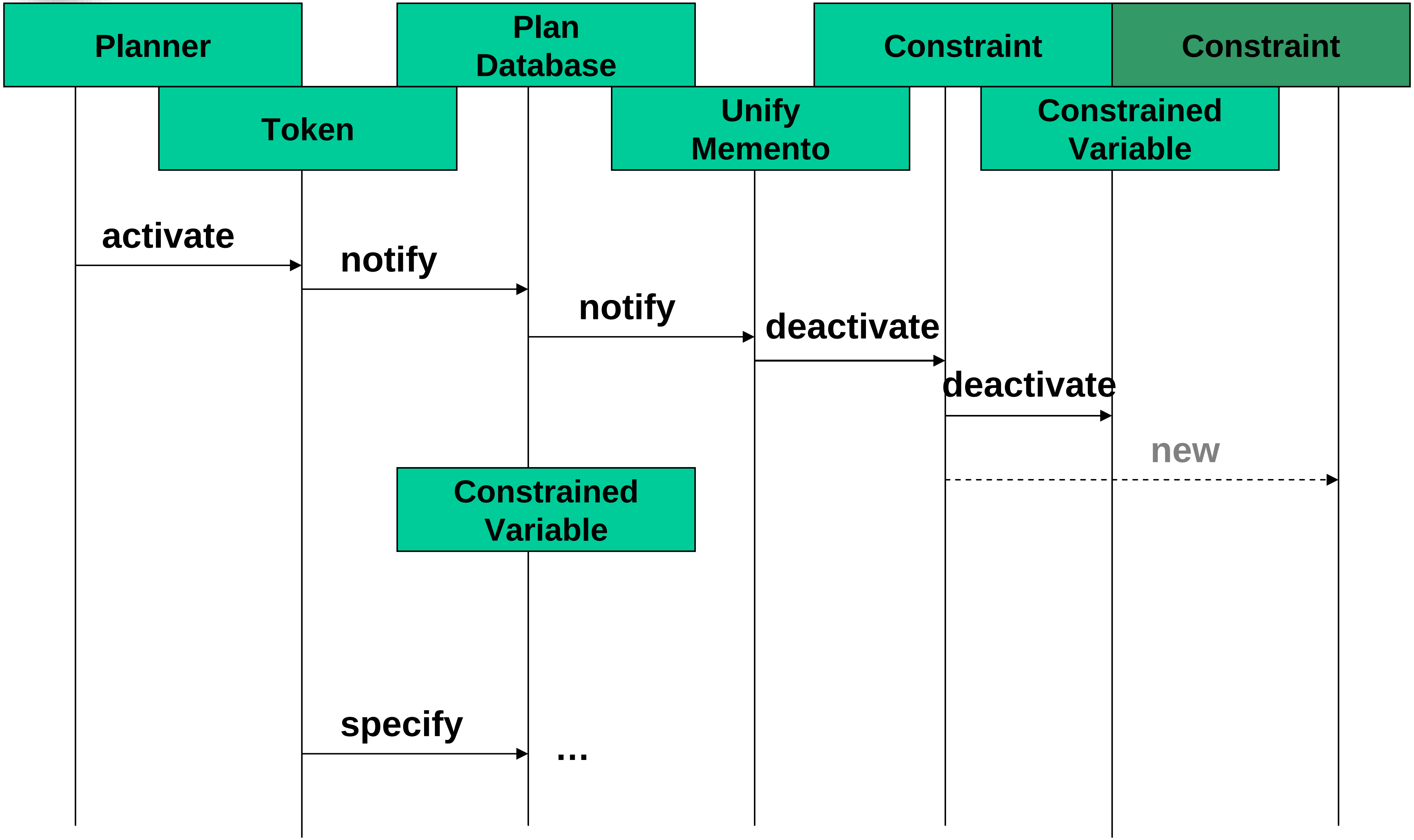
Token State Transition Model



Plan Database Framework
Activate: Collaboration Diagram



Plan Database Framework Merge: Collaboration Diagram



Decision Model

Variable Decisions (resolve unbound variables)

- specify(var,val) / reset(var)

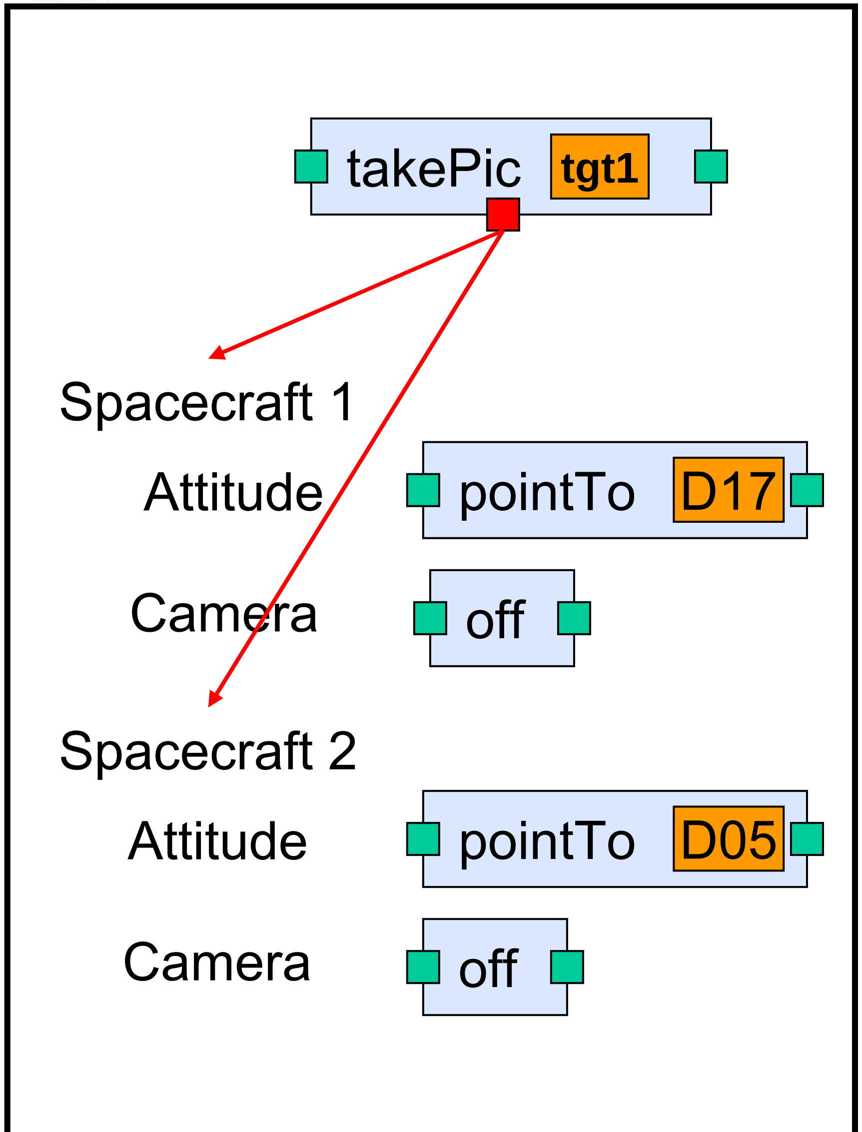
Token Decisions (resolve inactive tokens)

- activate(token) / deactivate(token)
- merge(inactiveToken,activeToken) / split(inactiveToken)
- reject(token) / reinstate(token)

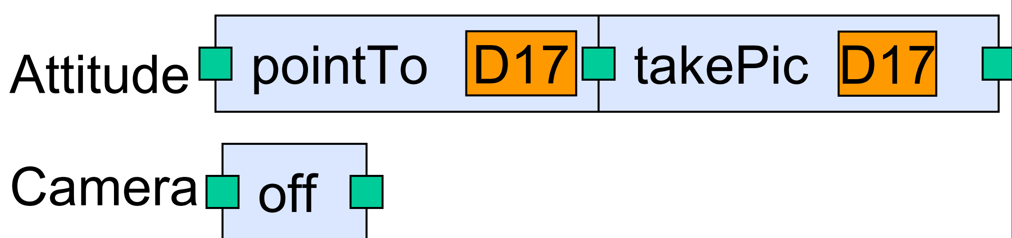
Object Decisions (resolve object with unordered tokens)

- constrain(object,predecessorToken,successorToken)
- free(object,predecessorToken,successorToken)

Multiple objects of same class



Constrain(spacecraft1,
pointTo(D17),
takePic(tg1))



Only active tokens can be constrained.

Plan Database Events

Nofications:

- TOKEN_ADDED / REMOVED
- OBJECT_ADDED / REMOVED
- TOKEN_ACTIVATED / DEACTIVATED
- TOKEN_MERGED / SPLIT
- TOKEN_REJECTED / REINSTATED
- TOKEN_CONSTRAINED / FREED
- TOKEN_ADDED_TO_OBJECT
- TOKEN_REMOVED_FROM_OBJECT

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ❑ NDDL – New Domain Description Language
- ❑ Assemblies
- ❑ PlanWorks
- ❑ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

New Domain Description Language (NDDL)

Main features:

- Procedural with Java-like syntax and semantics
- Allows file inclusion via #include directive
- Allows comments C++ style (// and /* */)
- Allows object composition
- Allows definition of static objects

NDDL Domain Descriptions:

- Must include basic NDDL constructs (Plasma.nddl, PlanConfig.nddl)
- Describes objects in the domain
- Describes predicates for each object (if appropriate)
- Describes configuration rules for predicates (e.g. precedences)

NDDL Problem Descriptions:

- Must include Planner Configuration (horizon and number of steps)
- Declares object instances in the domain
- Closes the Plan Database (announcing that it is ready to plan)
- Declares goals (rejectable or mandatory)

The Expressive Power of NDDL

Static objects

Temporally scoped predicates

Resources

Object composition

Object inheritance

Conditional subgoaling

Inifinite domains (limited capability)

Existential quantification

Universal quantification (over finite domains)

Define own constraints

Define own base types (via NDDL.cfg)

Modeling in NDDL

Static Objects:

```
class Location {
  int x;
  int y;
  string label;

  Location(int _x, int _y, string _label) {
    x = _x;
    y = _y;
    label = _label;
  }
}
```

Objects:

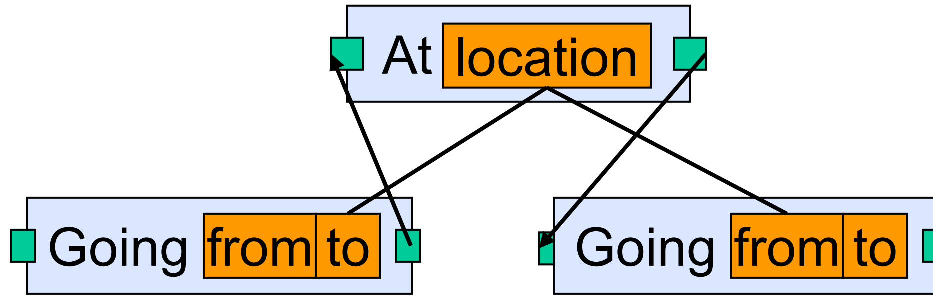
```
class Navigator {
  predicate At {
    Location location;
  }

  predicate Going {
    Location from;
    Location to;
    neq(from, to);
  }
}
```

NDDL Resources

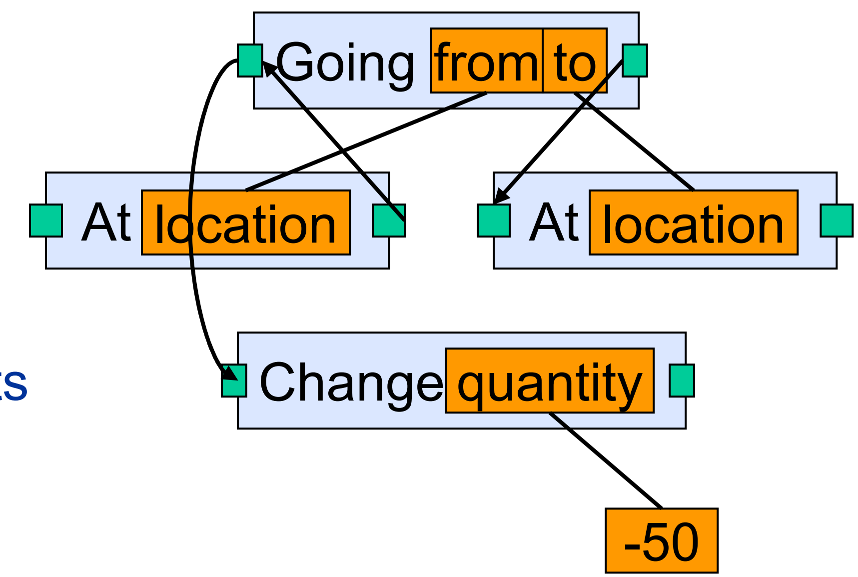
```
class Battery extends Resource {
  Battery(float ic, float ll_min, float ll_max){
   super(ic, ll_min, ll_max, 0.0, 0.0, MINUS_INFINITY, MINUS_INFINITY);
  }
}

class Resource {
   float initialCapacity;
   float levelLimitMin;
   float levelLimitMax;
   float productionRateMax;
   float productionMax;
   float consumptionRateMax;
   float consumptionMax;

 // The only predicate we allow
   predicate change{
      float quantity;
   }
}
```

NDDL Rules

```
Navigator::At {
 met_by(object.Going go_before);
 eq(go_before.to, location);
 meets(object.Going go_after);
 eq(go_after.from, location);
}
```




```
Navigator::Going {
 met_by(object.At at_before);
 eq(at_before.location, from);
 meets(object.At at_after);
 eq(at_after.location, to);
 starts(Battery.change tx);
 eq(tx.quantity, -50); // draws 50 units
}
```



NDDL Initial States

```
PlannerConfig plannerConfiguration = new PlannerConfig(0,100,50);

Location lander = new Location(0, 0, "lander");
Location rock1 = new Location(9, 9, "rock1");
Location rock2 = new Location(1, 6, "rock2");

// Allocate Rover with a battery
Battery battery = new Battery(1000.0, 0.0, 1000.0);
Rover spirit = new Rover(battery);

close(); // no more objects will be added

// Establish the initial position for spirit
goal(Navigator.At initialPosition);
initialPosition.start.specify(0); // Starts at the beginning of the horizon
initialPosition.location.specify(lander); // Initial position is lander
```

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

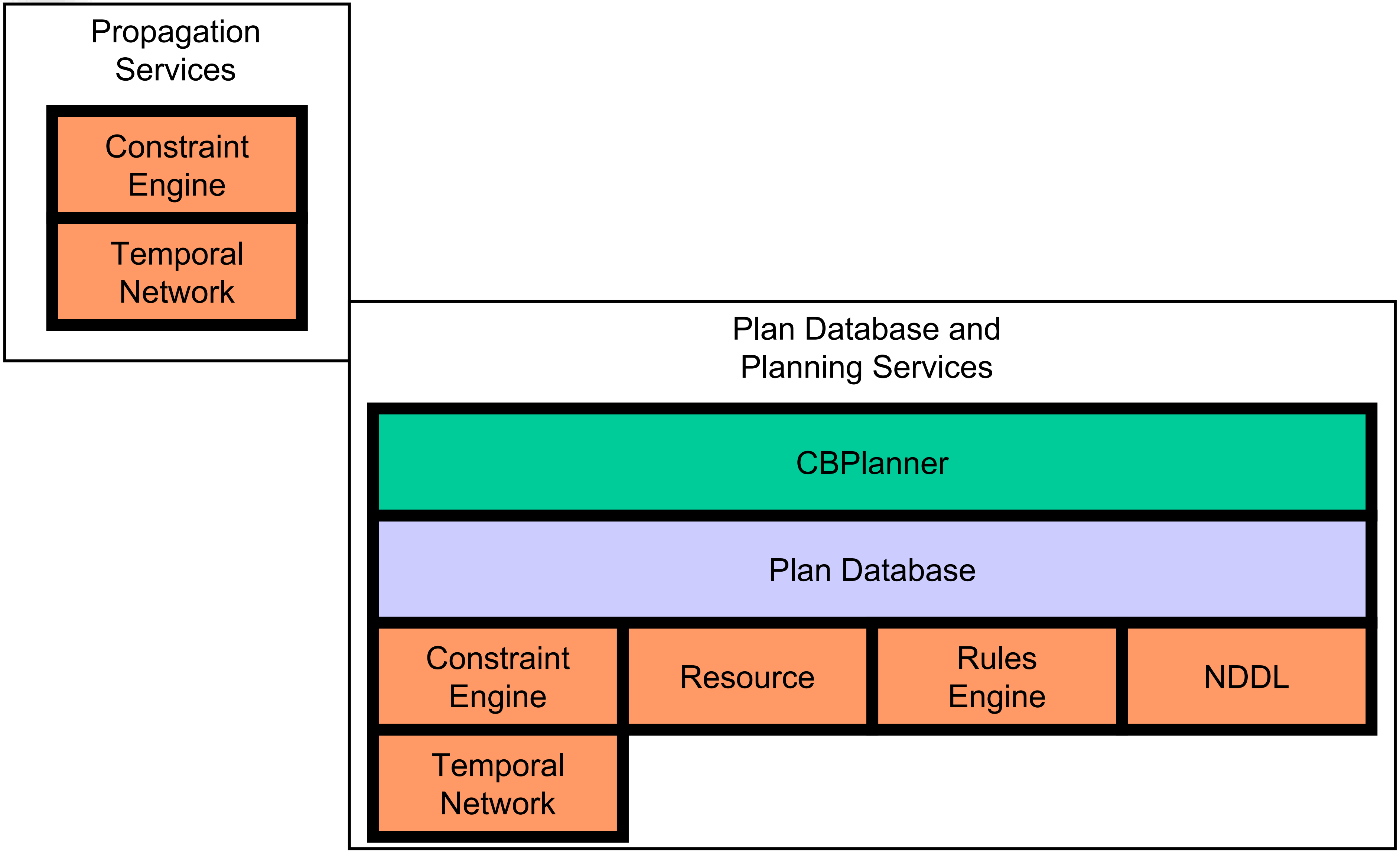
Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ❑ Assemblies
- ❑ PlanWorks
- ❑ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Assemblies: A Module View



Assemblies: Putting it all Together

Create services

- ConstaintEngine ce;

Register propagators

- new DefaultPropagator(LabelStr("Default"), ce);

Initialize NDDL and type factories

- initNDDL();

Register constraint types

- REGISTER_CONSTRAINT(EqualConstraint, "eq", "Default");

Create a planner

- CBPlanner planner(db, horizon);

Create a partial plan writer (if interfacing with PlanWorks)

- PlanWriter::PartialPlanWriter ppw(db, ce, re, planner);

Propagation Services Assembly

```
ConstraintEngine ce;
new DefaultPropagator(LabelStr("Default"), ce.getId());
new TemporalPropagator(LabelStr("Temporal"), ce.getId());

REGISTER_CONSTRAINT(TemporalDistanceConstraint, "StartEndDurationRelation", "Temporal");

--------------------------------------------------------------------------------------------------------------------

IntervalIntDomain domStart = IntervalIntDomain(0,10);
IntervalIntDomain domEnd = IntervalIntDomain(0,20);
IntervalIntDomain domDur = IntervalIntDomain(1,1000);

ConstrainedVariableId v1 = (new Variable<IntervalIntDomain> (ce.getId(), domStart, true, "v1"))->getId();
ConstrainedVariableId v2 = (new Variable<IntervalIntDomain> (ce.getId(), domDur, true, "v2"))->getId();
ConstrainedVariableId v3 = (new Variable<IntervalIntDomain> (ce.getId(), domEnd, true, "v3"))->getId();

std::vector<ConstrainedVariableId> temp;
temp.push_back(v1);
temp.push_back(v2);
temp.push_back(v3);
ConstraintId duration1 =
    ConstraintLibrary::createConstraint(LabelStr("StartEndDurationRelation"), ce.getId(), temp);

ce.propagate();
```

Plan Database and Planner Assembly

```
ConstraintEngine ce;
PlanDatabase db(ce, schema);
new DefaultPropagator(LabelStr("Default"), ce);
new TemporalPropagator(LabelStr("Temporal"), ce);
new ResourcePropagator(LabelStr("Resource"), ce, db);
Propagator temporalPropagator = ce.getPropagatorByName(LabelStr("Temporal"));
    db.setTemporalAdvisor((new STNTemporalAdvisor(temporalPropagator));
Rules Engine re(db);
initNDDL();
REGISTER_CONSTRAINT(EqualConstraint, "eq", "Default");
REGISTER_CONSTRAINT(ResourceConstraint, "ResourceRelation", "Resource");
REGISTER_CONSTRAINT(TemporalDistanceConstraint, "StartEndDurationRelation", "Temporal");
CBPlanner planner(db, horizon);
PlanWriter::PartialPlanWriter ppw(db, ce, re, planner);
std::list<ObjectId> configObjects;
m_planDatabase->getObjectsByType("PlannerConfig", configObjects); // Standard configuration class
ObjectId configSource = configObjects.front();
const std::vector<ConstrainedVariableId>& variables = configSource->getVariables();
ConstrainedVariableId horizonStart = variables[0];
ConstrainedVariableId horizonEnd = variables[1];
ConstrainedVariableId plannerSteps = variables[2];
int start = (int) horizonStart->baseDomain().getSingletonValue();
int end = (int) horizonEnd->baseDomain().getSingletonValue();
horizon.setHorizon(start, end);
int steps = (int) plannerSteps->baseDomain().getSingletonValue();
CBPlanner::Status res = m_planner->run(steps);
```

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ✓ Assemblies
- ❑ PlanWorks
- ❑ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Other Useful Modules

PlanWorks

- Plan visualization and query interface

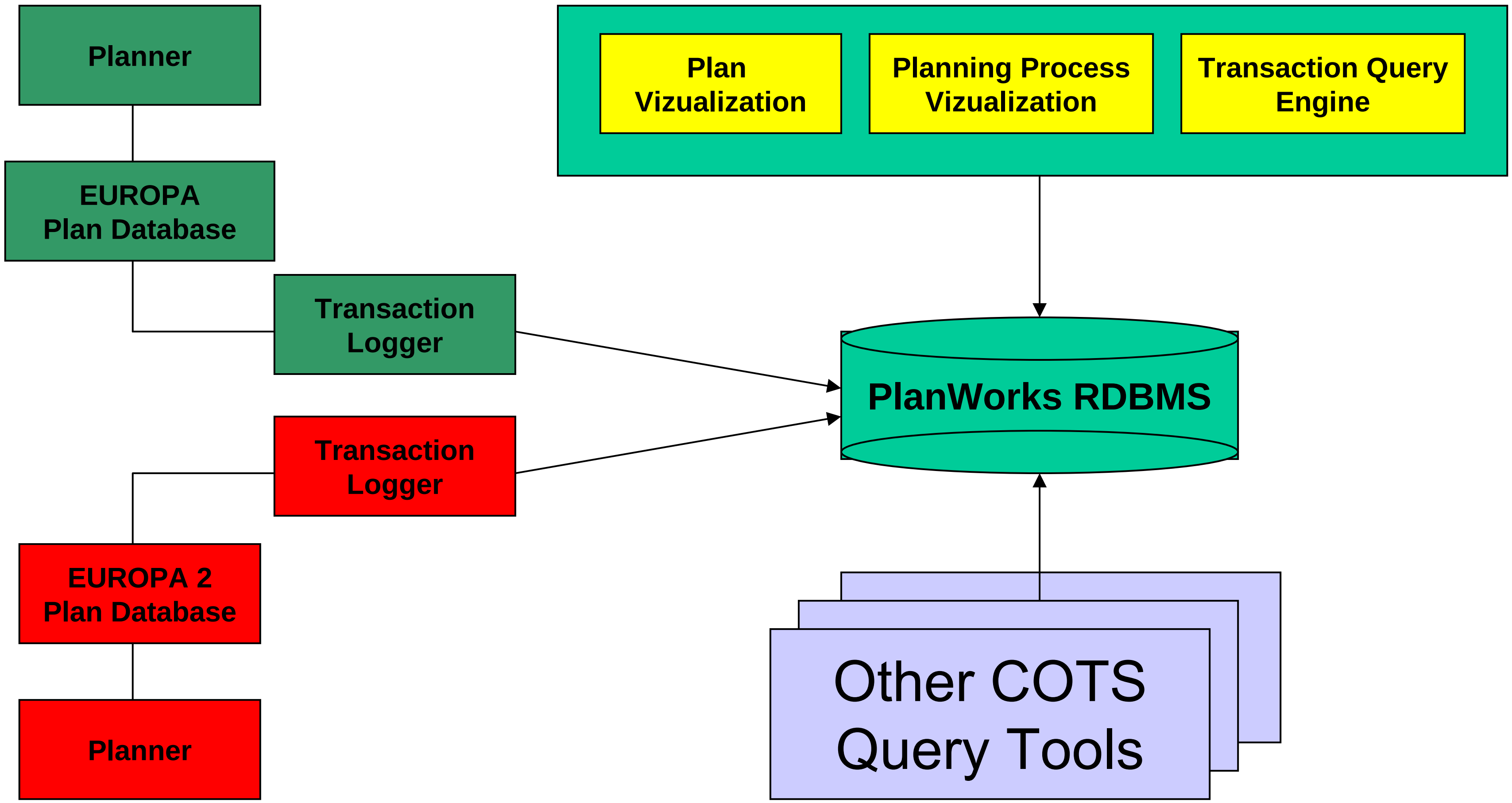
Aver

- Test language interpreter

HSTS

- Europa heuristics (HSTS heuristics – planned)
- Europa assembly (without resources)

PlanWorks Architecture



Aver

Language and Interpreter to check plan or planning behavior

Assertions:

- step specifications followed by a single boolean statement that asserts a property of the step
- composed of domain- and singleton-valued functions and a boolean operator ('<', '>', '>=', '<=', '=', '!=', 'in', 'out', 'intersects‘)

Example:

```
Test('Test',
  At first step : Count(Tokens()) = 11;
  At any step : Count(Transactions(type = 'RETRACTION')) > 1;
  At any step : Count(Transactions(type = 'ASSIGNMENT')) > 1;
  At any step : Count(Transactions(type = 'RESTRICTION')) > 1;
  At step = 87 : Count(Tokens()) = 65;
  At step = 87 : Count(Tokens(predicate = 'Target.Tracked')) == 4;
  At step = 87 : Count(Tokens(start >= 3)) > 0;
  At step = 87 : Count(Tokens(end = [11..500])) = 3;
  At step = 87 : Count(Tokens(predicate = 'Target.Tracked' variable(name = 'TYPE'
    value = 'FLUENT'))) = 4;
  At last step : Count(Tokens()) = 65;
);
```

HSTS

Objective: to support EUROPA migration

Provides:

- EUROPA Decision Ordering
- EUROPA Heuristics (in XML, automatic conversion)
- EUROPA NoBranch (as Condition, automatic conversion)
- HSTSAssembly
- EUROPA Semantics for LabelSet equivalence
- Sample of migrated domains

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ✓ Assemblies
- ✓ PlanWorks
- ✓ Aver
- ❑ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Extensions

- Creating your own constraint
- Creating your own specialized propagator
- Creating your own object model
- Creating your own subgoaling rules
- Creating specialized domains

Extensions
Creating Your Own Constraint

1. Declare your Class
2. Implement constructor
3. Implement handleExecute
4. Optionally, implement canIgnore (performance only)

```
SubsetOfConstraint::SubsetOfConstraint(
        const LabelStr& name,
        const LabelStr& propagatorName,
        const ConstraintEngineId& constraintEngine,
        const std::vector<ConstrainedVariableId>& variables) :
    Constraint(name, propagatorName, constraintEngine, variables),
    m_currentDomain(getCurrentDomain(variables[0])),
    m_superSetDomain(getCurrentDomain(variables[1])) {  }

void SubsetOfConstraint::handleExecute() {
    m_currentDomain.intersect(m_superSetDomain);
}

bool SubsetOfConstraint::canIgnore(const ConstrainedVariableId& variable, const
      DomainListener::ChangeType& changeType) {
   If (changeType == DomainListener::RESET || changeType == DomainListener::RELAXED)
      return false;
   else return true;
}
```

External Integrations

Event Listeners

- Constraint Engine
- Plan Database
- Rules Engine
- Decision Manager

External Data Integration:

- via XML transaction files – containing domain descriptions and plan database calls
- via the DbClient interface – useful to track the set of changes
- directly through the API
- via the planner control interface – wraps the model initialization and planner in a JNI object (can extend to plan database access)
- via xml query interface to mysql (would have to build xml interface)

Things I didn't tell you today

Error handling

- Enabling / disabling exceptions

Debug messages

- Configuring debug messages

How to create your own:

- Rules
- Decision points and choices
- Specialized propagator
- Specialized domains
- Custom objects and tokens

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ✓ Assemblies
- ✓ PlanWorks
- ✓ Aver
- ✓ Extensions

Part III

- ❑ Build your own model
- ❑ Visualize it in PlanWorks

Building Your Own Project

1. Ensure Europa 2 is running:
   - Cd PLASMA root
   - Jam tests

2. Create your own project:
   a) Cd PLASMA root
   b) ./makeproject UserGuideRover
   c) Cd ../UserGuideRover
   d) Jam UserGuideRover
   e) View RUN_UserGuideRover-planner_g_rt.UserGuideRover-initial-state.xml

UserGuideRover-model.nddl

```
#include "../PLASMA/NDDL/core/Plasma.nddl"
#include "../PLASMA/NDDL/core/PlannerConfig.nddl"

/**
 * @brief Place holder class with a single predicate
 */
class YourObject extends Timeline {
 predicate helloWorld{} /*!< Predicate with no arguments */
}

/**
 * @brief A simple rule to force a repeated cycle
 */
YourObject::helloWorld{
 eq(duration, 10);
 meets (object.helloWorld);
 met_by(object.helloWorld);
}
```

UserGuideRover-initial-state.nddl

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

// Create a planner configuration instance in PLASMA.
// Horizon Start, Horizon End, MaxPlannerSteps
// new: changed horizon from 1000 to 200.
PlannerConfig plannerConfiguration = new PlannerConfig(0, 200, 500);

// Sample object
YourObject object = new YourObject();

// Close the the PLASMA Database - no more objects can be created.
close();

// Now place your goals here.
goal(YourObject.helloWorld initialToken);
initialToken.start.specify(0); // Starts at beginning of the horizon

// The planner should take it form here!
```

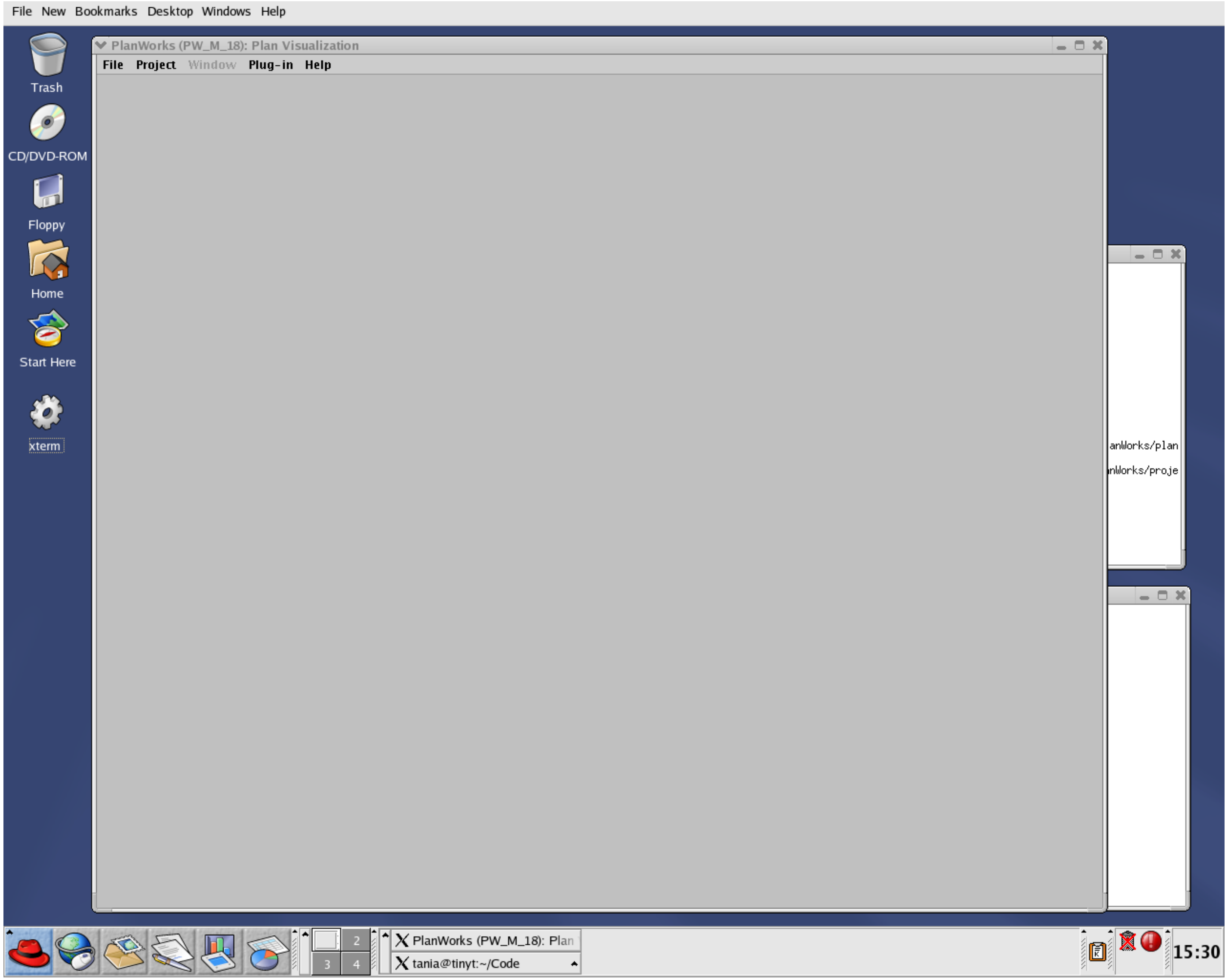
UserGuide-Main.cc

```
#include "Nddl.hh" /*!< Includes protypes required to load a model */
#include "StandardAssembly.hh" /*!< For using a standard EUROPA Assembly */
#include "Constraints.hh"
#include "ConstraintLibrary.hh"
#include "MyConstraint.hh"

using namespace EUROPA;

int main(int argc, const char ** argv){
  if (argc != 2) {
    std::cerr << "Must provide initial transactions file." << std::endl;
    return -1;
  }
  const char* txSource = argv[1];
  StandardAssembly::initialize();
  SchemaId schema = NDDL::loadSchema();
  // Encapsulate allocation so that they go out of scope before calling terminate
  {
    StandardAssembly assembly(schema);
    assembly.plan(txSource);
  }
  StandardAssembly::terminate();

  std::cout << "Finished\n";
}
```

Visualizing with PlanWorks

1. Modify PlanWorks.cfg
2. Ensure PlanWorks is running:
   - Cd PlanWorks root
   - Ant

Plan Works Initial



UserGuideRover In PlanWorks

- Create a Project
- Point to the plans directory inside UserGuideRover
- Load a planning sequence
- Display the Timeline View
- Scroll one by one
- Point out differences between steps

Add Parameters and Constraints

```
#include "../PLASMA/NDDL/core/Plasma.nddl"
#include "../PLASMA/NDDL/core/PlannerConfig.nddl"

enum MyEnum {one, two, three}

class YourObject extends Timeline {
 predicate helloWorld {
   int theInt = [1 3];
   MyEnum theEnum;
 }
}

YourObject::helloWorld{
 eq(duration, 10);
// meets(object.helloWorld);
// met_by(object.helloWorld);
 meets(YourObject.helloWorld prev);
 neq(prev.theInt,theInt);
 met_by(YourObject.helloWorld next);
 neq(next.theInt,theInt);
 neq(prev.theInt,next.theInt);
}
```

Create and Add Your Own Constraint

- Create your own constraint header and implementation
- Modify the main program to register the constraint and include ConstraintLibrary.hh and Constraint.hh.
- Modify the nddl model file to reference the constraint.
- Modify the Jamfile to compile and link the constraint implementation

MyConstraint Header

```
#ifndef _H_MyConstraint
#define _H_MyConstraint

#include "ConstraintEngineDefs.hh"
#include "Constraint.hh"

namespace EUROPA {

  class MyConstraint : public Constraint {
  public:
    MyConstraint(const LabelStr& name,
                 const LabelStr& propagatorName,
                 const ConstraintEngineId& constraintEngine,
                 const std::vector<ConstrainedVariableId>& variables);

    void handleExecute();

  private:
    static const int X = 0;
    static const int Y = 1;
  };

}
#endif
```

MyConstraint Implementation

```
#include "MyConstraint.hh"
#include "Variable.hh"
#include "IntervalIntDomain.hh"
#include "EnumeratedDomain.hh"
#include "Error.hh"

namespace EUROPA {

  MyConstraint::MyConstraint(const LabelStr& name, const LabelStr& propagatorName, const ConstraintEngineId& constraintEngine, const
          std::vector<ConstrainedVariableId>& variables) : Constraint(name, propagatorName, constraintEngine, variables) {
    check_error (variables.size() == 2);
    check_error (getCurrentDomain(variables[X]).getType() == AbstractDomain::INT_INTERVAL);
    check_error (getCurrentDomain(variables[Y]).getType() == AbstractDomain::SYMBOL_ENUMERATION);
  }

  void MyConstraint::handleExecute() {
    AbstractDomain& domx = getCurrentDomain(m_variables[X]);
    AbstractDomain& domy = getCurrentDomain(m_variables[Y]);

    if (domx.isOpen() || domy.isOpen()) return;
    check_error(!domx.isEmpty());   check_error(!domy.isEmpty());
    std::list<double> values;
    domx.getValues(values);
    for (std::list<double>::iterator it = values.begin(); it != values.end(); ++it) {
      int value = (int)(*it);
      switch (value) {
      case 1:  if (!domy.isMember(LabelStr("one").getKey()))
                 domy.empty();
               break;
      case 2: if (!domy.isMember(LabelStr("two").getKey()))
                 domy.empty();
              break;
      case 3:  if (!domy.isMember(LabelStr("three").getKey()))
                 domy.empty();
              break;
      default: check_error(ALWAYS_FAIL);
              break;
      }
    }
  }

}
```

UserGuideRover-main.cc

```
#include "Nddl.hh" /*!< Includes protypes required to load a model */
#include "StandardAssembly.hh" /*!< For using a standard EUROPA Assembly */
#include "Constraints.hh"
#include "ConstraintLibrary.hh"
#include "MyConstraint.hh"

using namespace EUROPA;

int main(int argc, const char ** argv){
  if (argc != 2) {
    std::cerr << "Must provide initial transactions file." << std::endl;
    return -1;
  }
  const char* txSource = argv[1];
  StandardAssembly::initialize();
  SchemaId schema = NDDL::loadSchema();
  // Encapsulate allocation so that they go out of scope before calling terminate
  {
    REGISTER_CONSTRAINT(MyConstraint, "myConstraint", "Default");
    StandardAssembly assembly(schema);
    assembly.plan(txSource);
}
  }

  // Terminate the library
  StandardAssembly::terminate();

  std::cout << "Finished\n";
}
```

UserGuideRover-model.nddl

```
#include "../PLASMA/NDDL/core/Plasma.nddl"
#include "../PLASMA/NDDL/core/PlannerConfig.nddl"

enum MyEnum {one, two, three}

class YourObject extends Timeline {
 predicate helloWorld {
   int theInt = [1 3];
   MyEnum theEnum;
   myConstraint(theInt,theEnum);
 }
}

YourObject::helloWorld{
 eq(duration, 10);
 meets(YourObject.helloWorld prev);
 neq(prev.theInt,theInt);
 met_by(YourObject.helloWorld next);
 neq(next.theInt,theInt);
 neq(prev.theInt,next.theInt);
}
```

Jamfile

```
SubDir UGR ;

if ! $(UGR_READY) {
 # Create a build target to run a problem
 RunProblem UGR : UGR-initial-state.nddl : UGR-planner ;

 # Create a build target for the planner executable with the
   given model.
 NddlMain UGR-planner : MyConstraint.cc UGR-Main.cc :
   UGR-model3.nddl : NDDL : UGR-planner ;
} # UGR_READY
```

Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ✓ Assemblies
- ✓ PlanWorks
- ✓ Aver
- ✓ Extensions

Part III

- ✓ Build your own model
- ❑ Visualize it in PlanWorks

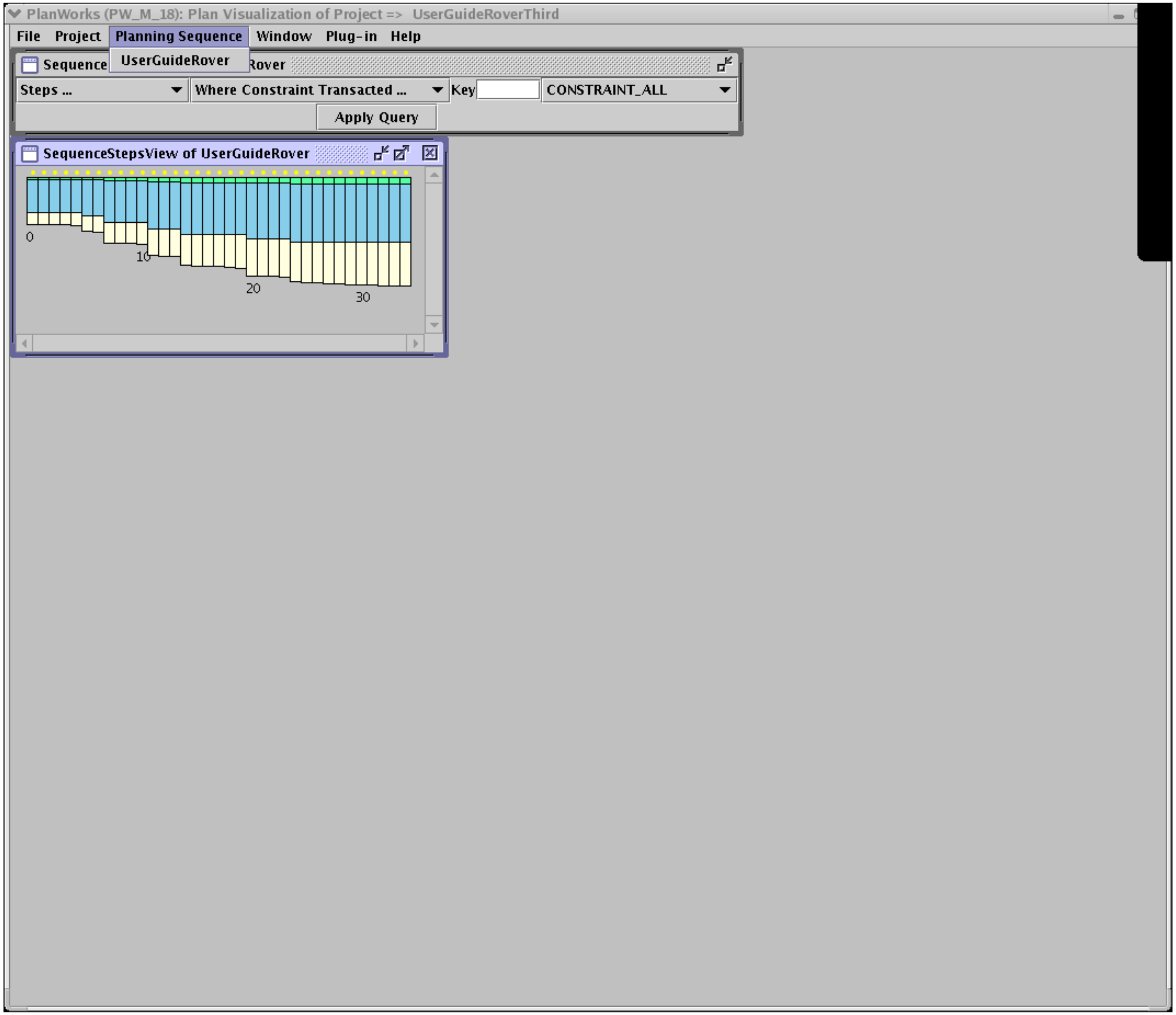
Using a More Complicated Model

Let us use the model in the User Guide.

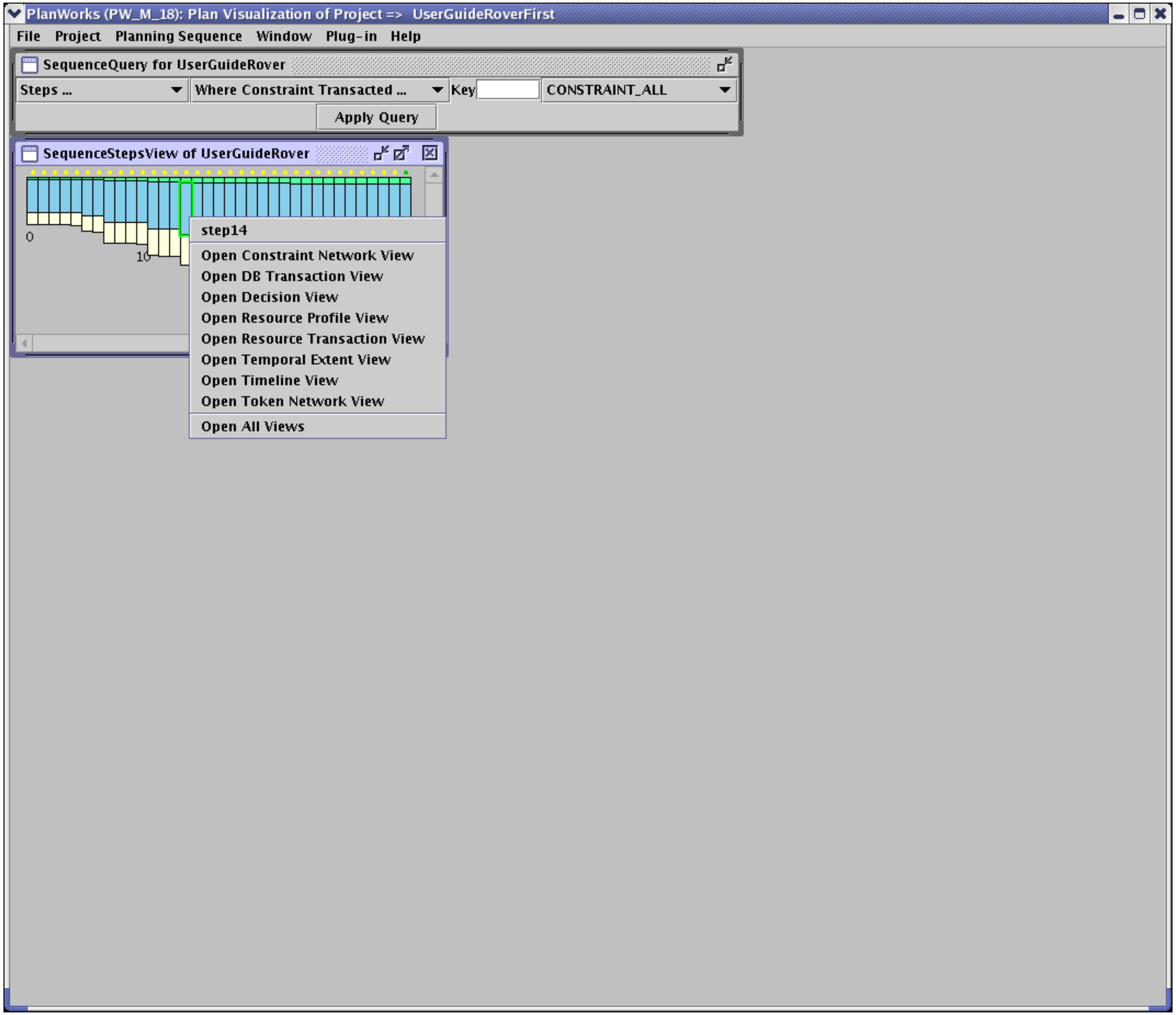
a) cp ../PLASMA/documentation/UserGuideRover*.nddl .
b) modify Jamfile to pick up new files or rename the files
c) jam UserGuideRover or the default target
d) verify ./plans directory has been created and that it has a datafile

Load new sequence in PlanWorks

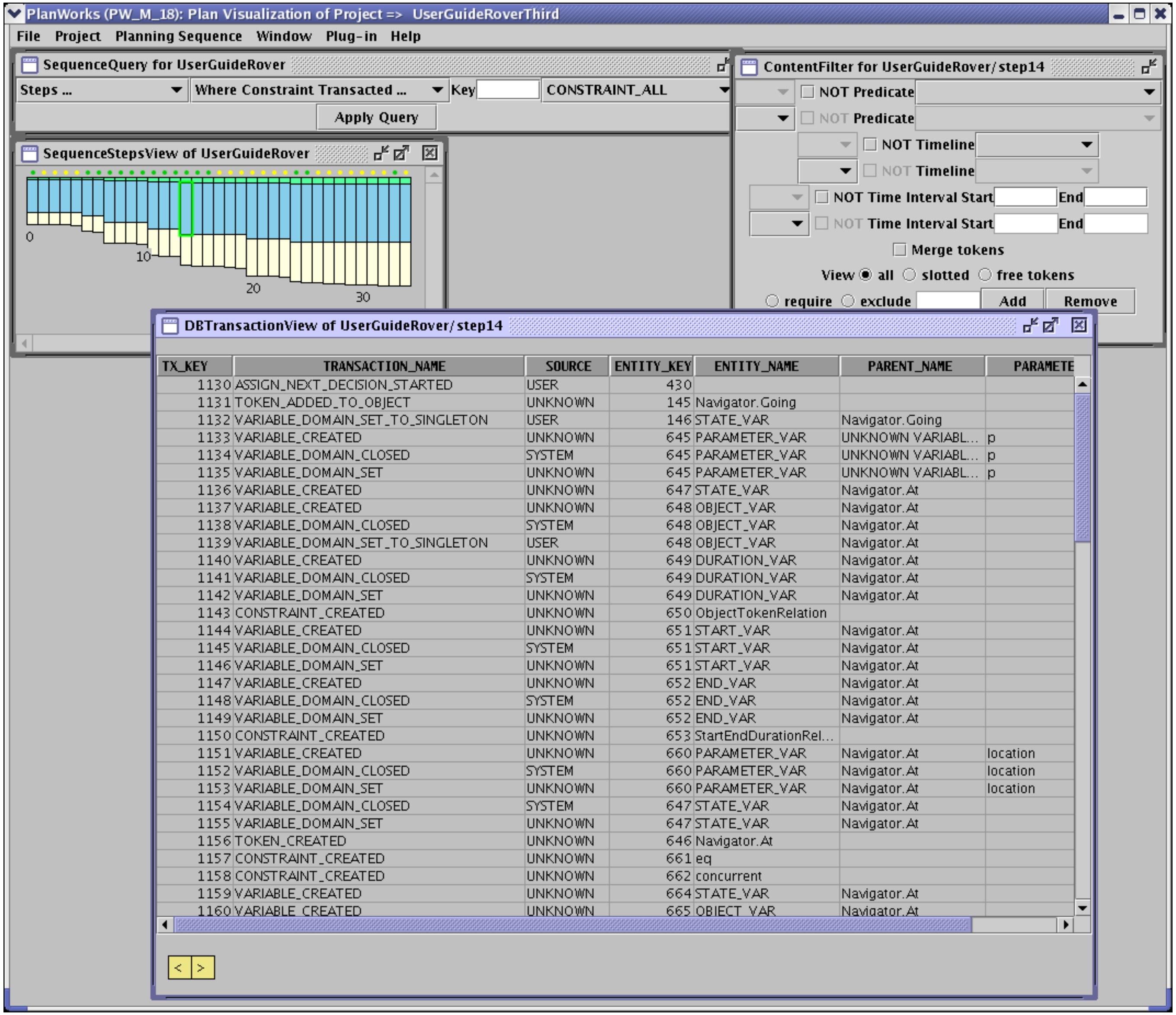
PlanWorks Planning Sequence



Plan Works Step Menu View

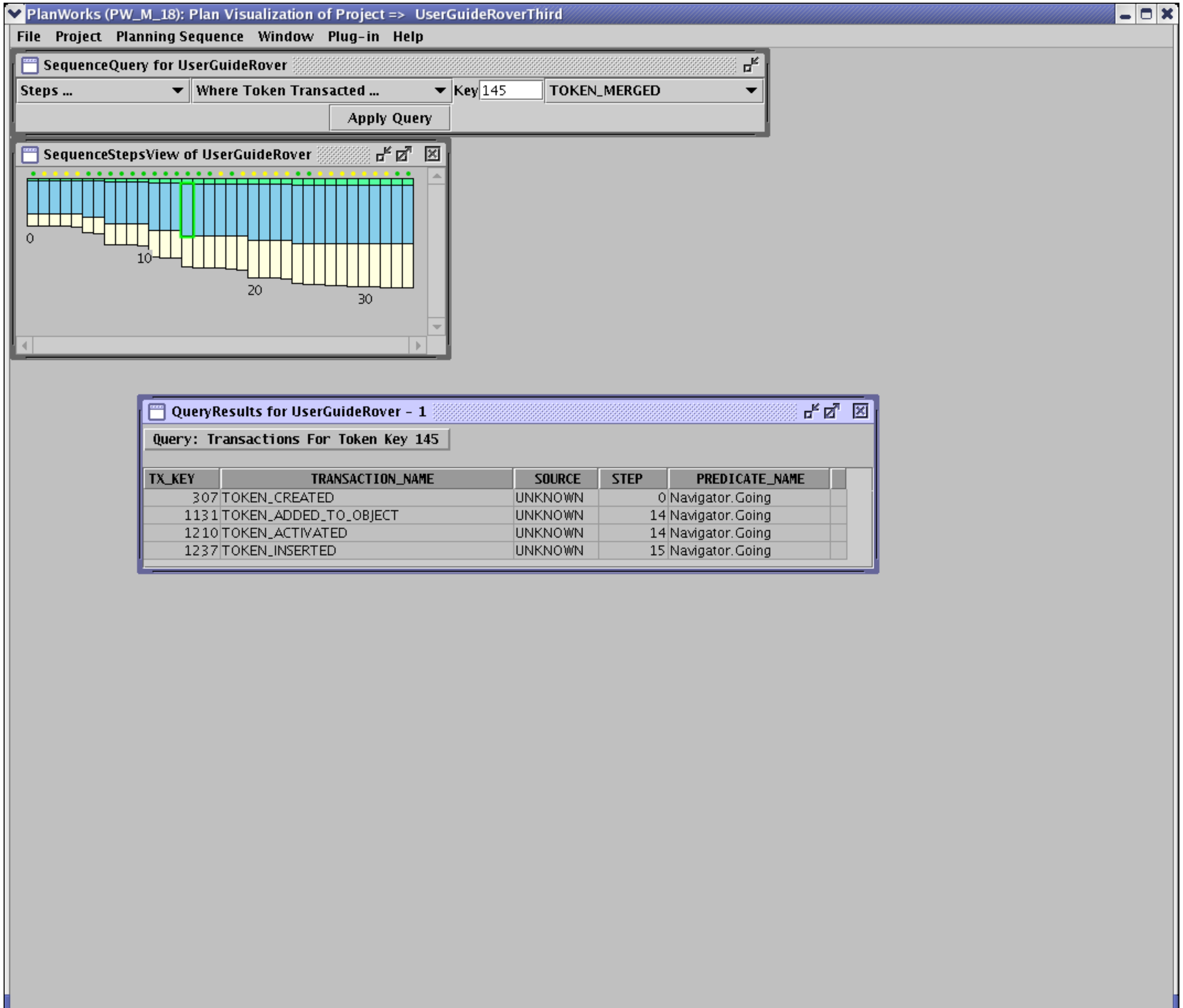


PlanWorks Transaction View

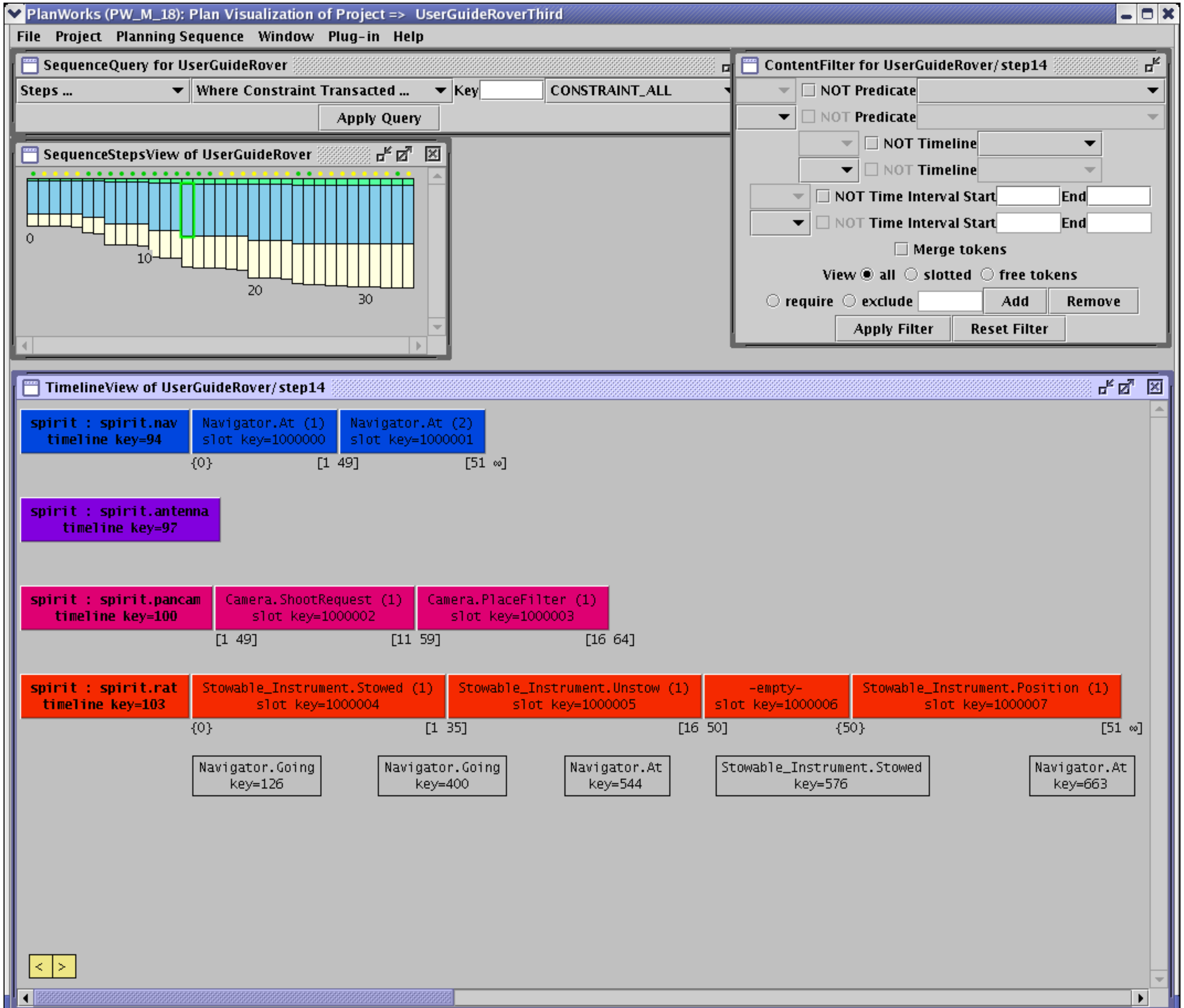


TX_KEY	TRANSACTION_NAME	SOURCE	ENTITY_KEY	ENTITY_NAME	PARENT_NAME	PARAMETE
1130	ASSIGN_NEXT_DECISION_STARTED	USER	430			
1131	TOKEN_ADDED_TO_OBJECT	UNKNOWN	145	Navigator.Going		
1132	VARIABLE_DOMAIN_SET_TO_SINGLETON	USER	146	STATE_VAR	Navigator.Going	
1133	VARIABLE_CREATED	UNKNOWN	645	PARAMETER_VAR	UNKNOWN VARIABL...	p
1134	VARIABLE_DOMAIN_CLOSED	SYSTEM	645	PARAMETER_VAR	UNKNOWN VARIABL...	p
1135	VARIABLE_DOMAIN_SET	UNKNOWN	645	PARAMETER_VAR	UNKNOWN VARIABL...	p
1136	VARIABLE_CREATED	UNKNOWN	647	STATE_VAR	Navigator.At	
1137	VARIABLE_CREATED	UNKNOWN	648	OBJECT_VAR	Navigator.At	
1138	VARIABLE_DOMAIN_CLOSED	SYSTEM	648	OBJECT_VAR	Navigator.At	
1139	VARIABLE_DOMAIN_SET_TO_SINGLETON	USER	648	OBJECT_VAR	Navigator.At	
1140	VARIABLE_CREATED	UNKNOWN	649	DURATION_VAR	Navigator.At	
1141	VARIABLE_DOMAIN_CLOSED	SYSTEM	649	DURATION_VAR	Navigator.At	
1142	VARIABLE_DOMAIN_SET	UNKNOWN	649	DURATION_VAR	Navigator.At	
1143	CONSTRAINT_CREATED	UNKNOWN	650	ObjectTokenRelation		
1144	VARIABLE_CREATED	UNKNOWN	651	START_VAR	Navigator.At	
1145	VARIABLE_DOMAIN_CLOSED	SYSTEM	651	START_VAR	Navigator.At	
1146	VARIABLE_DOMAIN_SET	UNKNOWN	651	START_VAR	Navigator.At	
1147	VARIABLE_CREATED	UNKNOWN	652	END_VAR	Navigator.At	
1148	VARIABLE_DOMAIN_CLOSED	SYSTEM	652	END_VAR	Navigator.At	
1149	VARIABLE_DOMAIN_SET	UNKNOWN	652	END_VAR	Navigator.At	
1150	CONSTRAINT_CREATED	UNKNOWN	653	StartEndDurationRel...		
1151	VARIABLE_CREATED	UNKNOWN	660	PARAMETER_VAR	Navigator.At	location
1152	VARIABLE_DOMAIN_CLOSED	SYSTEM	660	PARAMETER_VAR	Navigator.At	location
1153	VARIABLE_DOMAIN_SET	UNKNOWN	660	PARAMETER_VAR	Navigator.At	location
1154	VARIABLE_DOMAIN_CLOSED	SYSTEM	647	STATE_VAR	Navigator.At	
1155	VARIABLE_DOMAIN_SET	UNKNOWN	647	STATE_VAR	Navigator.At	
1156	TOKEN_CREATED	UNKNOWN	646	Navigator.At		
1157	CONSTRAINT_CREATED	UNKNOWN	661	eq		
1158	CONSTRAINT_CREATED	UNKNOWN	662	concurrent		
1159	VARIABLE_CREATED	UNKNOWN	664	STATE_VAR	Navigator.At	
1160	VARIABLE_CREATED	UNKNOWN	665	OBJECT_VAR	Navigator.At	

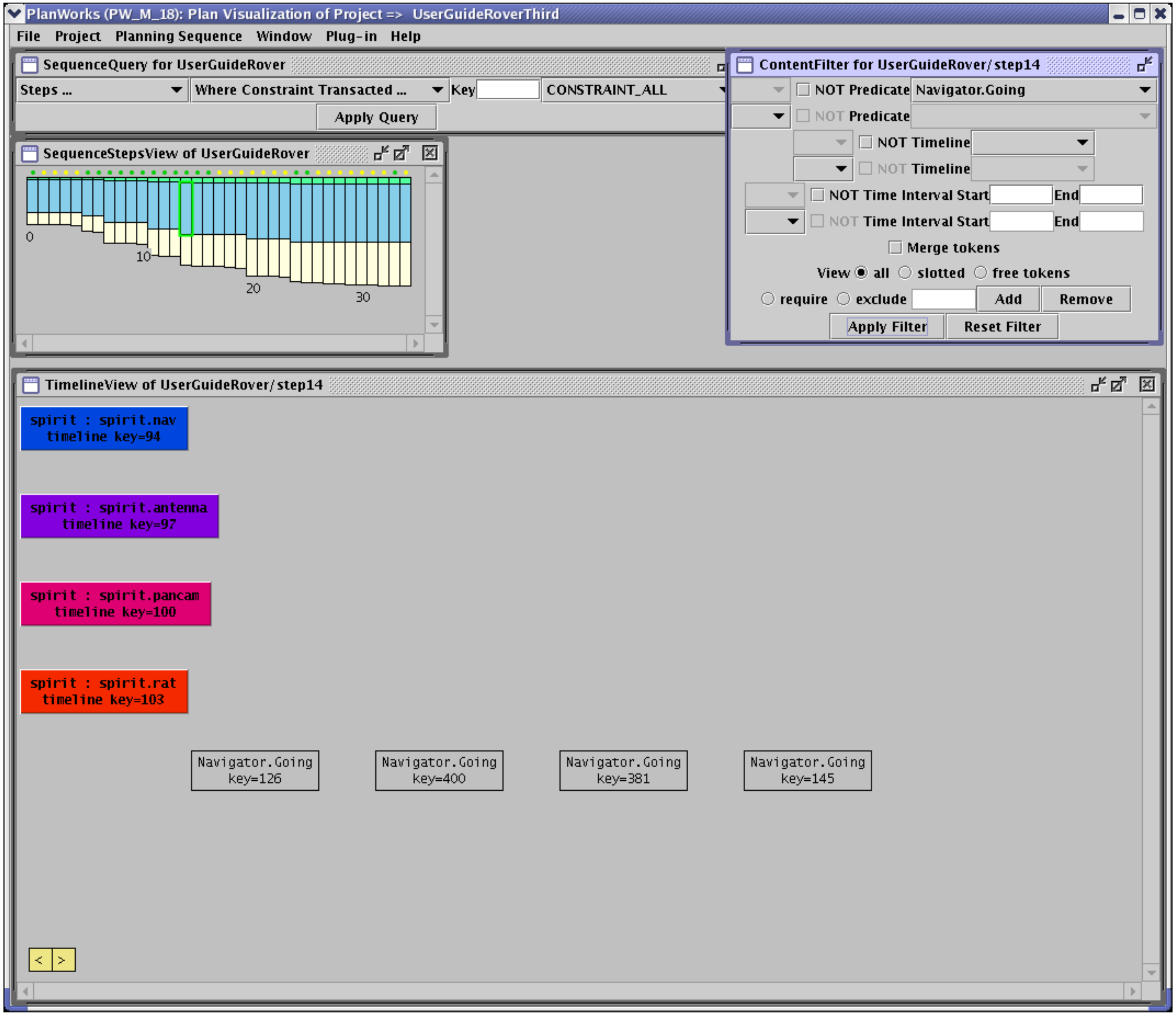
PlanWorks Token Query



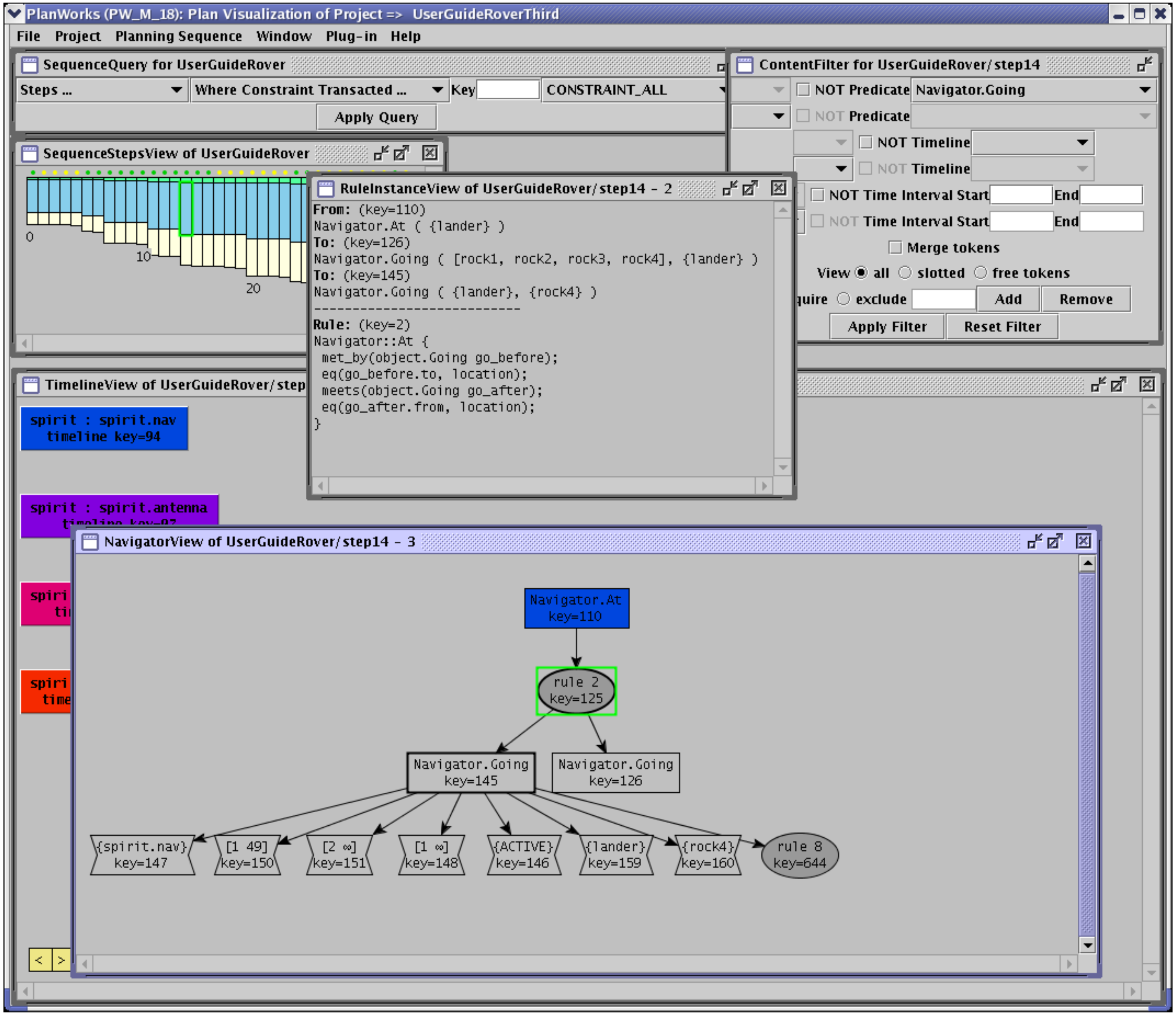
Plan Works Timeline View



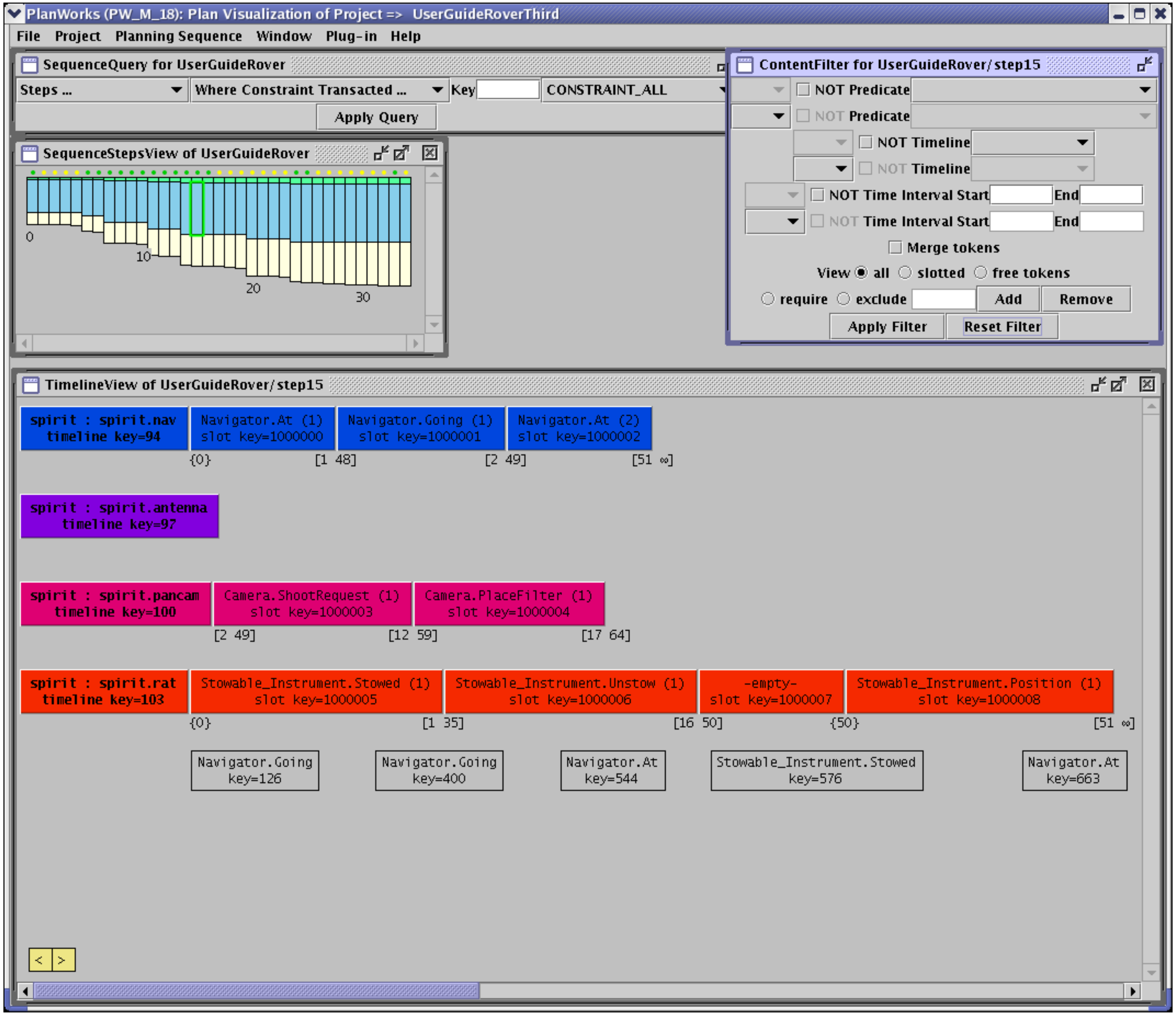
PlanWorks Timeline View Filtered



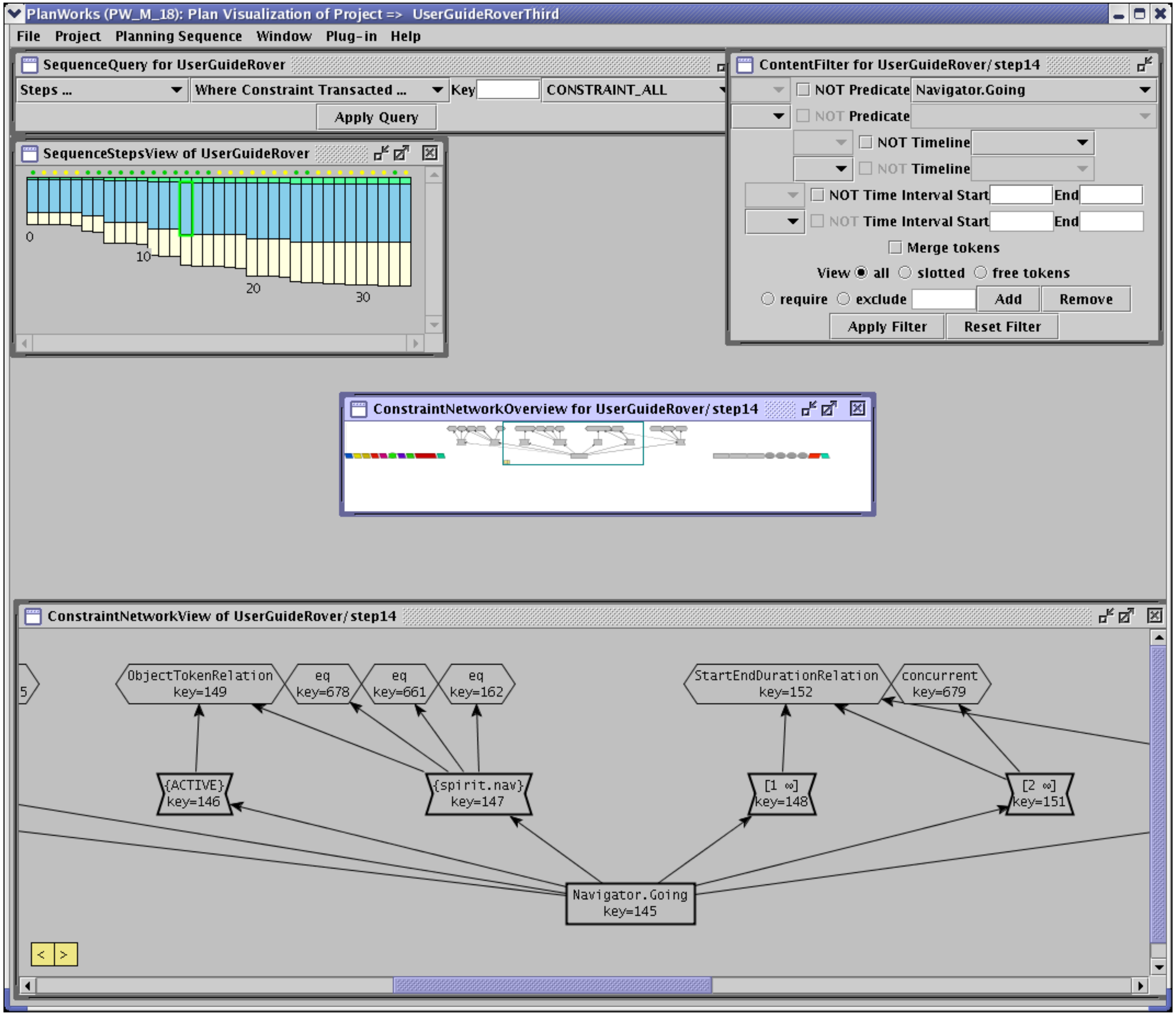
PlanWorks Rule Instance View



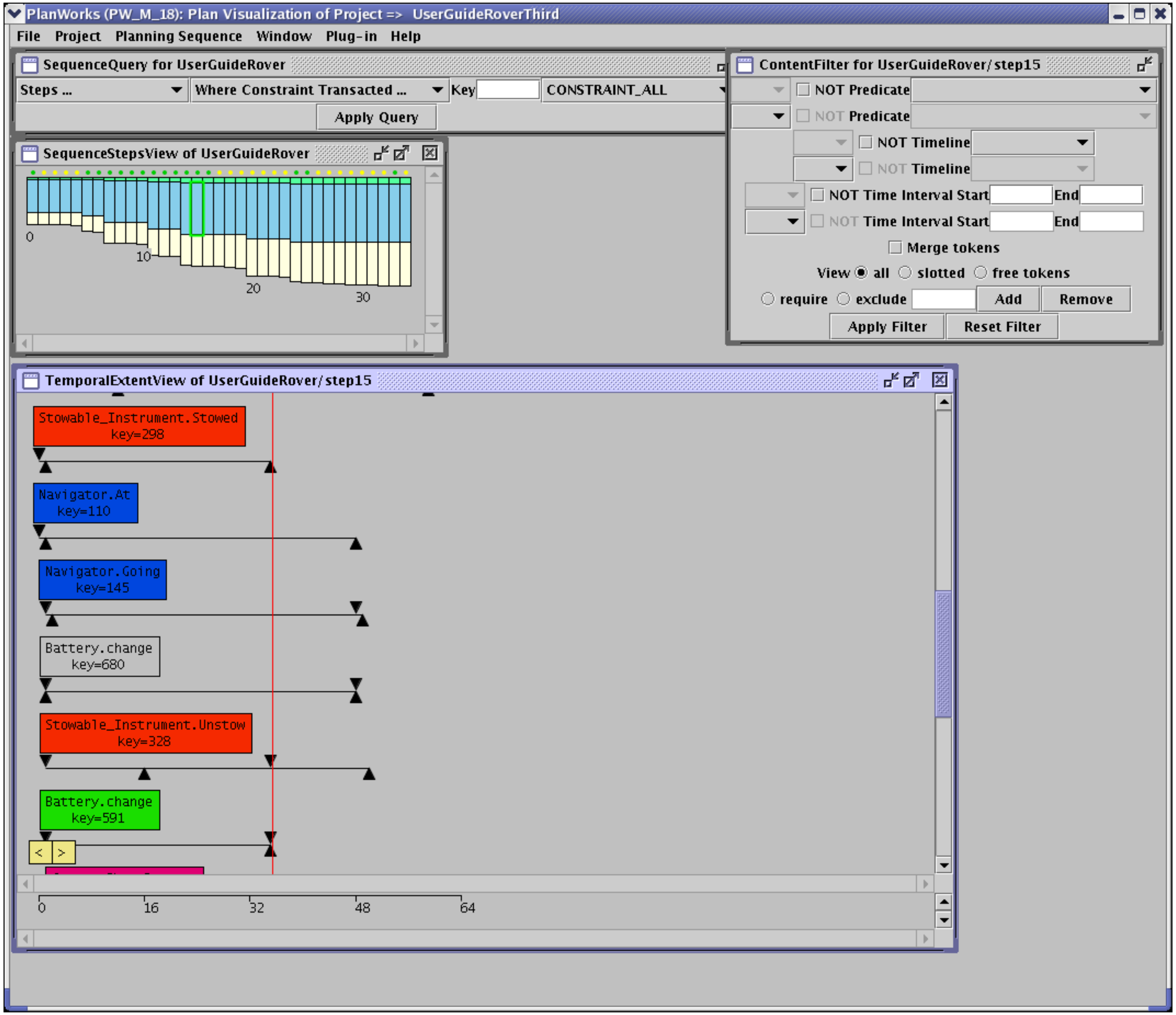
PlanWorks Timeline View Forward



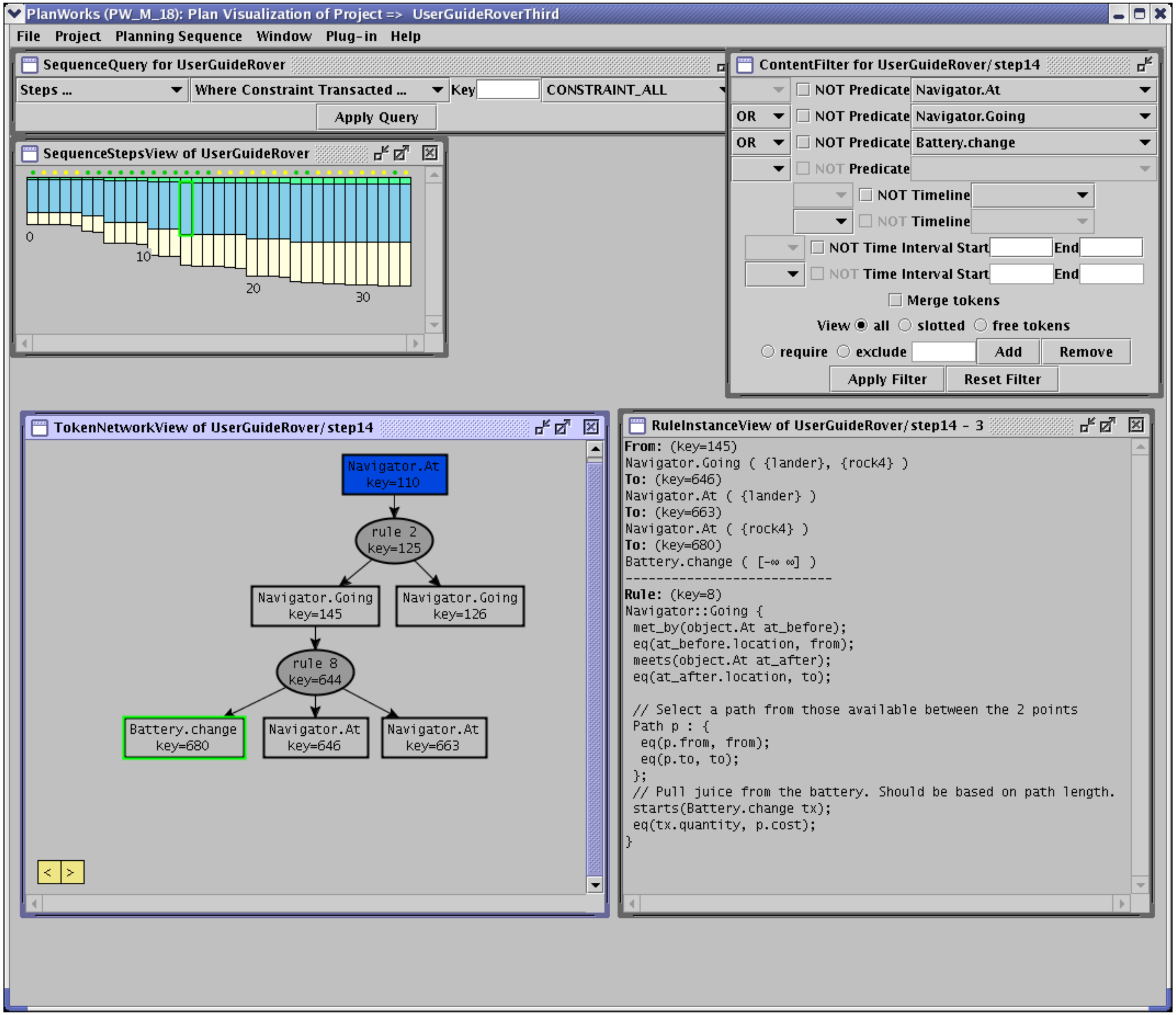
PlanWorks Constraint Network View



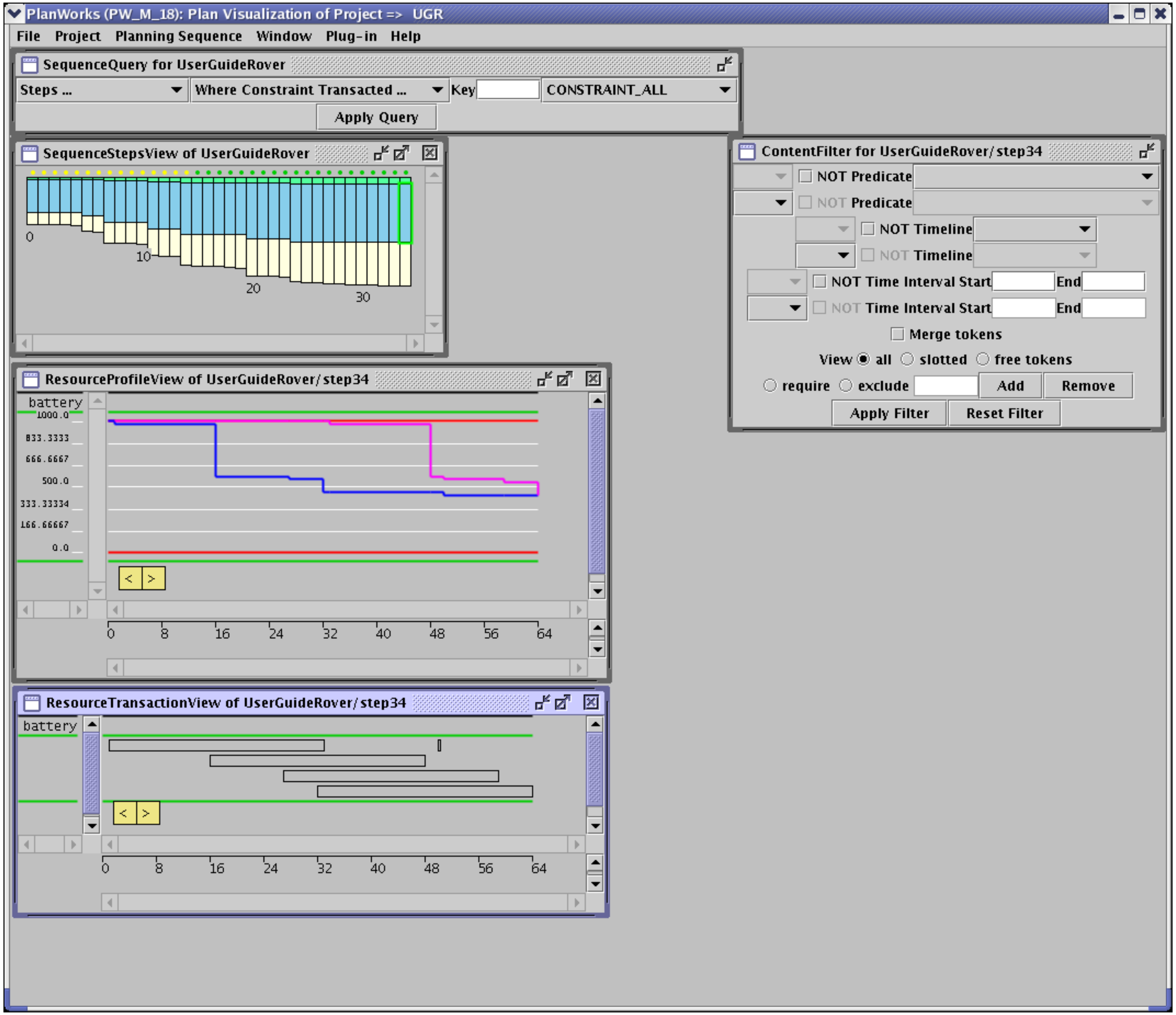
PlanWorks Temporal Extent



PlanWorks Token Network View



PlanWorks Resource View



Overview

Part I

- ✓ Motivation
- ✓ Background on Constraint-Based Planning

Part II

- ✓ Architecture
- ✓ NDDL – New Domain Description Language
- ✓ Assemblies
- ✓ PlanWorks
- ✓ Aver
- ✓ Extensions

Part III

- ✓ Build your own model
- ✓ Visualize it in PlanWorks

Objectives

To Understand:

- Constraint-Based Planning Paradigm
- EUROPA 2 and its Use Cases
- How to Create Your Own Project
- How to Generate a Plan and Visualize it in PlanWorks
- Possible Extensions and Create Your Own Constraint
- Modeling Features and Their Use