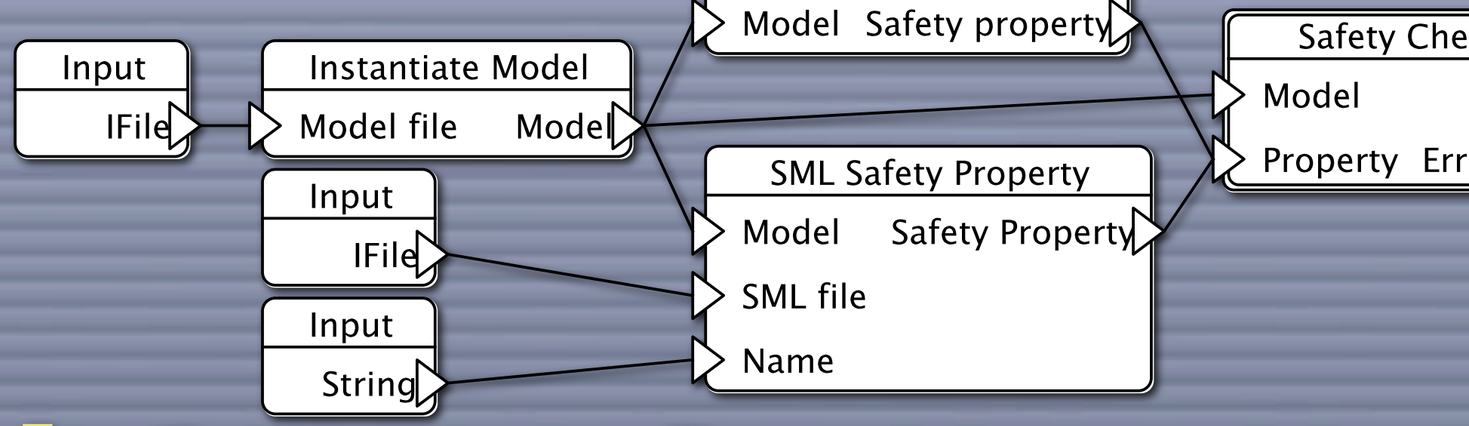
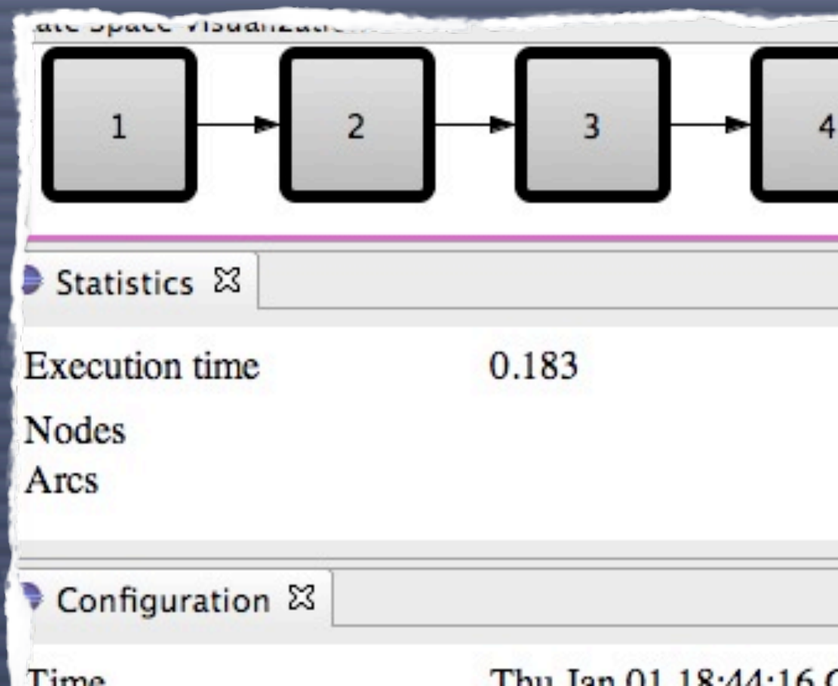


Advanced State Space Methods and ASAP: Extending ASAP

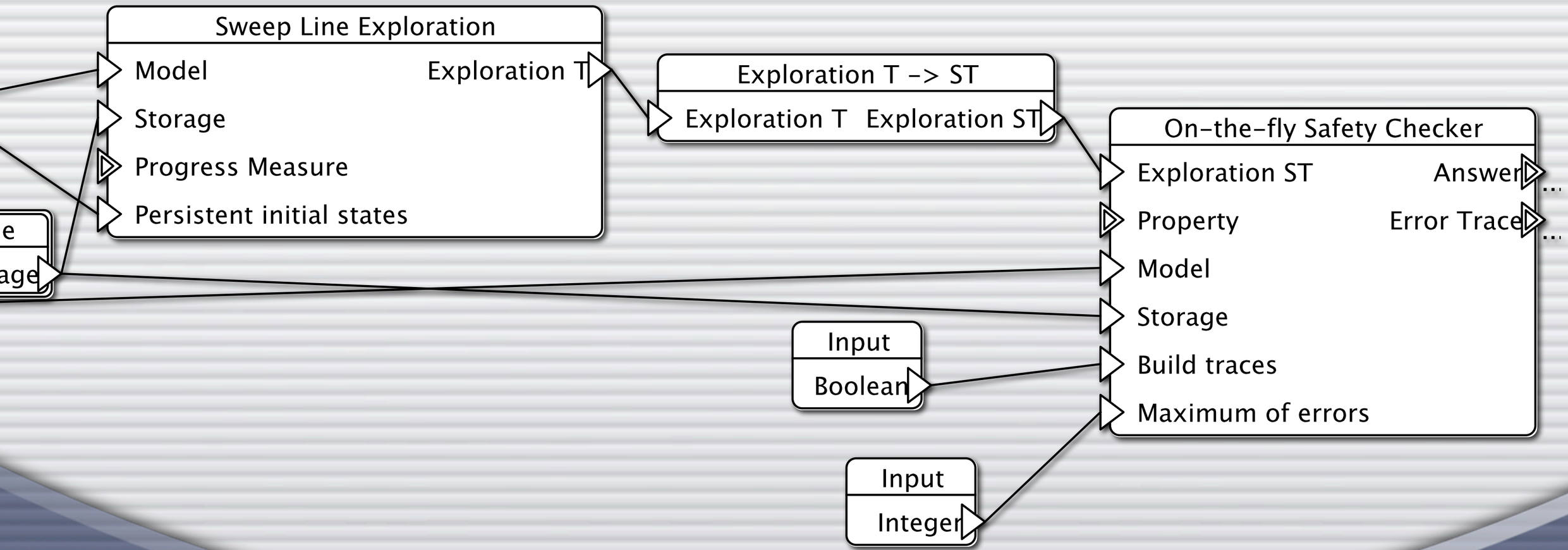
Michael Westergaard
Department of Computer Science
Aarhus University
mw@cs.au.dk

$V := \{ s_0 \}$
$W := \{ s_0 \}$
while $W \neq \emptyset$ **do**
Select an $s \in W$
$W := W \setminus \{ s \}$
if $\neg I(s)$ **then**
    return false
for all t, s' **such that** $s \rightarrow^t s'$ **do**
    if $s' \notin V$ **then**
        $V := V \cup \{ s' \}$
        $W := W \cup \{ s' \}$
return true



Requirements for Extendability

- It must be easy to add new features/ methods to ASAP
- New features must be completely integrated and feel like “native” features
- It should be possible to mix and match new features – even from different vendors



Example:
The Sweep-line Method

Example: The Sweep-line Method

- The sweep-line is defined completely outside of the ASAP main application (proof-of-concept, eat-your-own-dog-food, …)
- Yet…
  - We can add it in the JoSEL editor
  - We can use it with the safety-checker
  - We can combine it with hash compaction
  - We can create progress measures as easily as we create safety properties
  - The progress measure shows up in the report

Overview

- Adding new methods to the GUI
  - Eclipse's plug-in system
- Adding new methods to the engine
- Extending JoSEL
- Adding entries to the report
- Briefly: ACCESS/CPN

Basically, this is Easy!

- ASAP is an Eclipse Rich Client application, so we have access to Eclipse's plug-in mechanism
- This allows us to easily add new GUI elements (like the wizard for creating progress measures)
- This allows us to specify new points where the application can be extended

Eclipse's Plug-in System: Plug-ins

- **Plug-ins**: a program unit that provides a bounded functionality (e.g., the sweep-line method)
- **Dependencies**: a plug-in may (acyclically) depend on one or more other plug-ins (e.g., the sweep-line method depends on the generic state-space tool in ASAP)

Eclipse's Plug-in System: Extensions

- A plug-in may define zero or more **extension points** (e.g., new entries to add to the right-click menu in the index)
- An extension point can define any number of **details** (like the class implementing the wizard or when the menu entry should be enabled)
- An **extension point** provides an implementation of an extension point

A (Flawed) Analogy

- Think of plug-ins as pages of a CPN model
- Think of extension-points as input port places (that can be assigned zero, one, or more times)
- Think of extensions as socket places and port/socket-assignments

Plug-ins in ASAP

- ASAP uses mostly standard or slightly specialized standard components
- These thus get a lot of extensibility automatically
  - E.g., adding an entry to the right-click menu of the queries folder for creating a progress measure

Interfaces

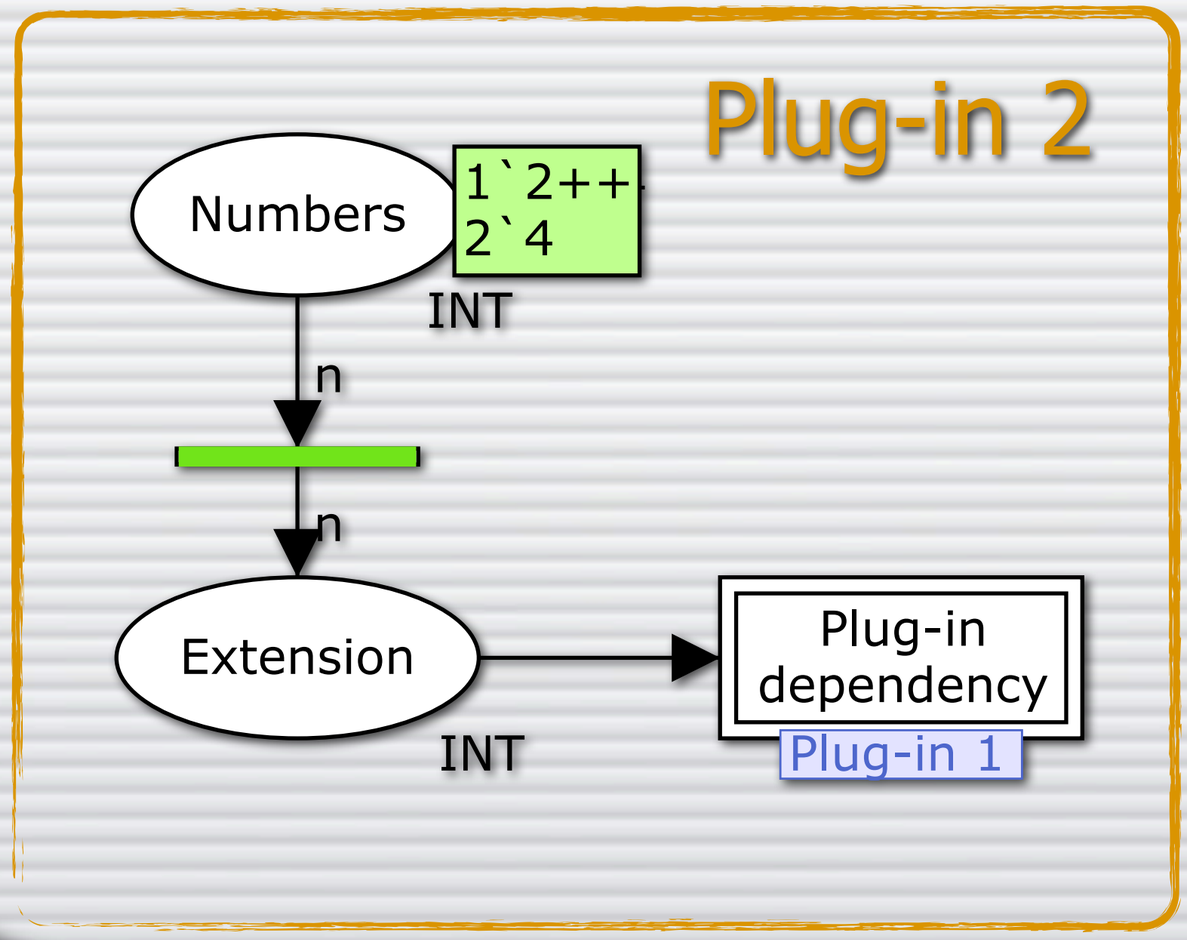
- In order to make this possible we need to adhere to the principle

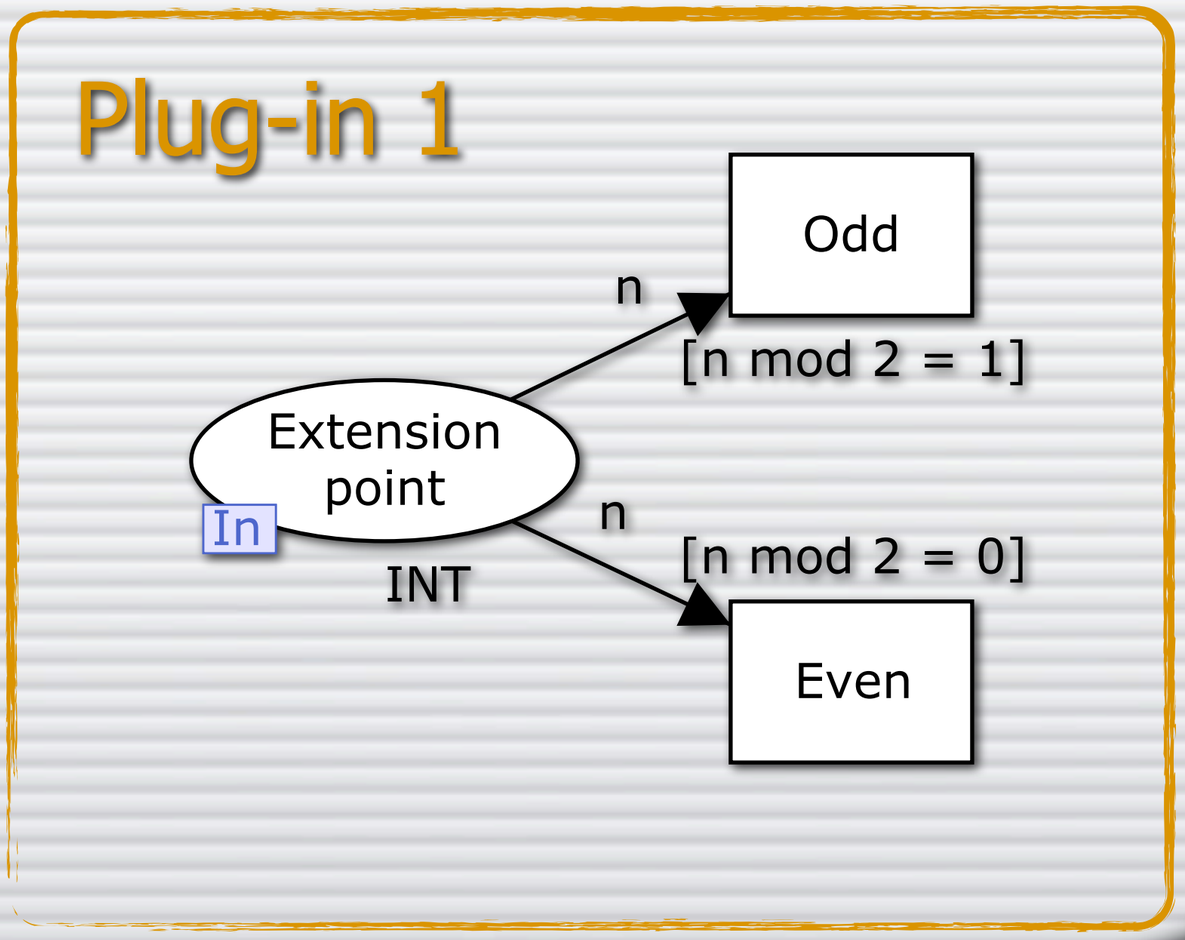
Program to the interface, not the implementation

- A plug-in defining an extension point describes which values are allowed, including which interfaces they must implement
  - The plug-in only has access to implementations via the interface

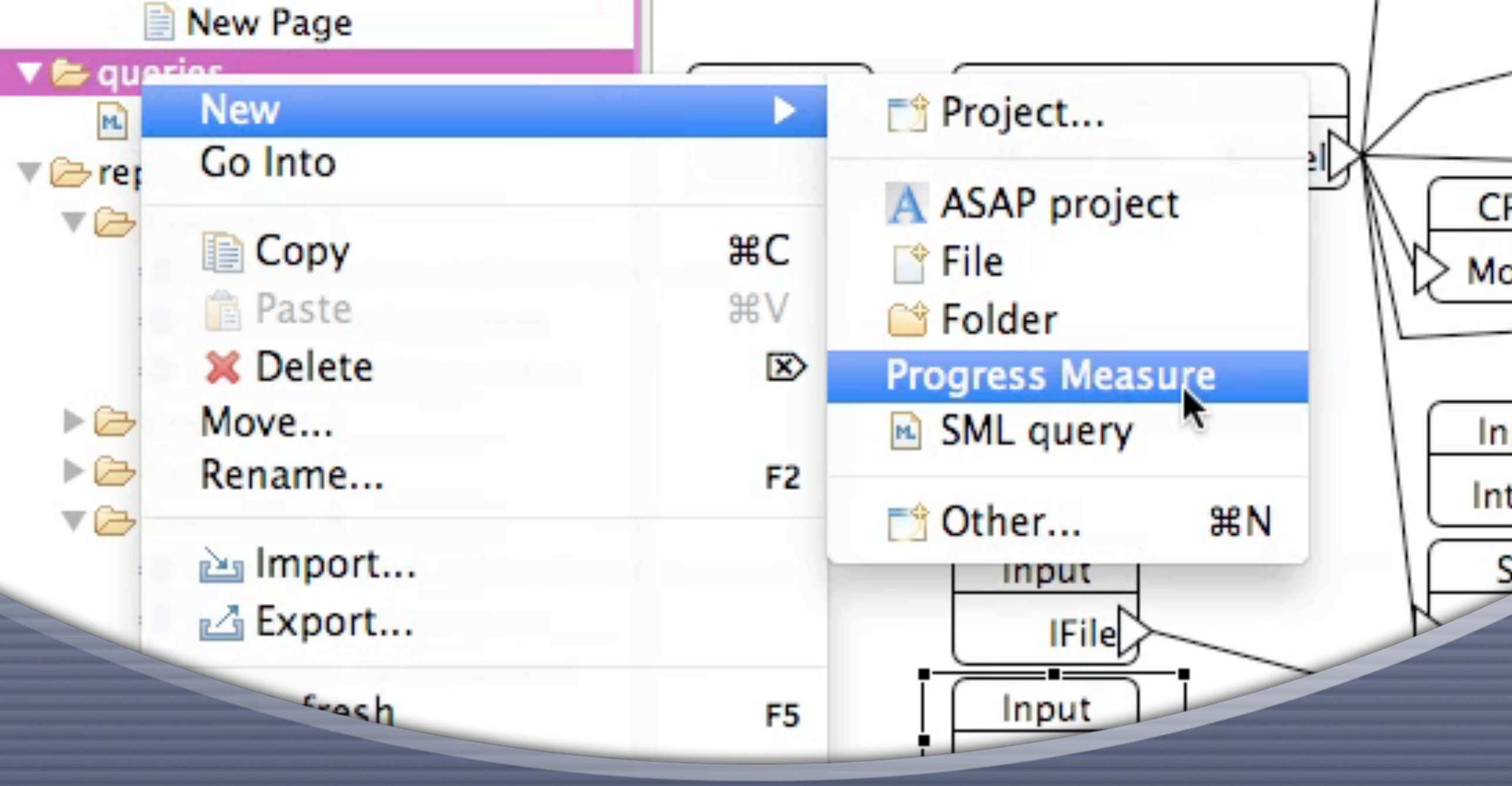
Expanding on the (Flawed) Analogy

- The interface of a port place is its type
  - The sub-page can make no assumptions other than token produced on the port place are of the correct type
  - In order to make a port/socket-assignment we must promise to only produce tokens of that type (the socket must have the same type, even though we can guarantee that we only produce tokens of a sub-type)





Two Plug-ins with the (Flawed) Analogy

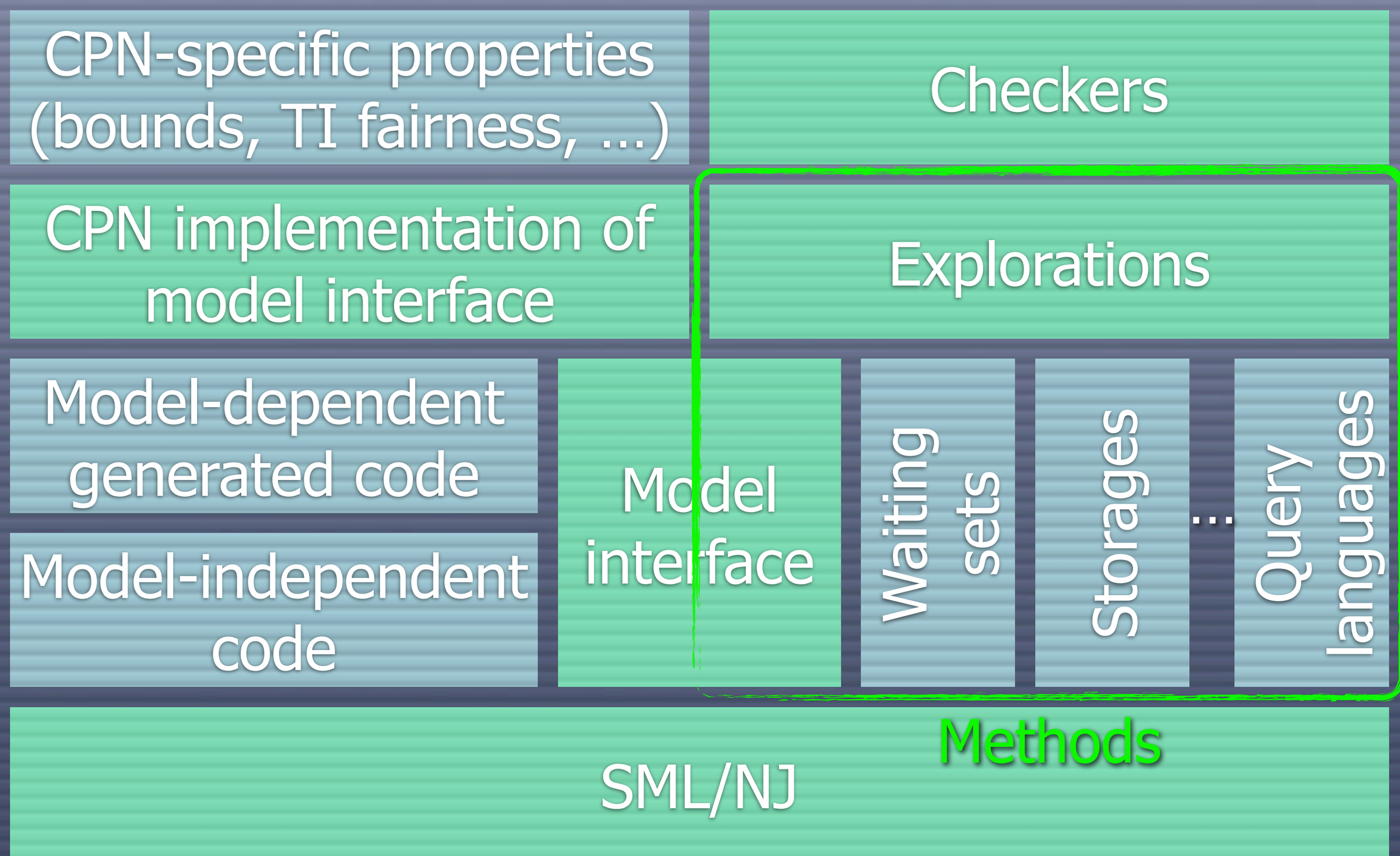


Example:
Adding Entry for PM

Demo: Adding Entry for PM (12)

- Show navigatorContent extension
  - Wizard & Enablement
- Show newWizards extension

State-space Tool of ASAP



Adding New Methods

- The state space engine of ASAP also introduces strict interfaces
  - Model, Storage, WaitingSet, Exploration (actually several explorations)
- Adding a new method should depend on these interfaces and implement interfaces (or define new interfaces and implement them)

Interfaces in SML

- SML uses **signatures** for interfaces
- Modules implementing interfaces are called **structures** or **functors**
  - Functors can explicitly depend on other structures and should be preferred over structures

```
fun sweep (□, storage, sVal, aVal) = (storage, sVal, aVal)
  | sweep (roots, storage, sVal, aVal) = let
      val _ = sweepHook (List.map #1 roots, storage)
      (*
       *  put root states into the queue (the toDel bit is set to false)
       *)
      val queue =
        List.foldl
            (fn ((s, id, trace), q) =>
              PQ.insert ((s, id, getProgress s, false, trace), q))
            (PQ.mkQueue (fn (_, _, prog, _, _) => prog)) roots
      val (toDel, roots, storage, sVal, aVal) =
        PQ.fold handleState queue (NONE, □, storage, sVal, aVal)
val (storage, sVal, aVal) =
    toDel
                    ids) => gc (storage, ids
```

Example: Sweep-line Exploration

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage                : REMOVE_STORAGE
structure Model                  : MODEL
structure Measure                : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

We require:
a boolean

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

We require:
a boolean
a storage

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

We require:
a boolean
a storage
a model

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

We require:
a boolean
a storage
a model
a progress measure

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

We require:
a boolean
a storage
a model
a progress measure

We provide:
a sweep-line exploration

Sweep-line Exploration Interfaces

- The “PROGRESS_MEASSURE” signature is defined by the sweep-line plug-in (and only applicable for the sweep-line method)
- The “SWEEP_LINE_EXPLORATION” signature is defined by the sweep-line plug-in, but extends the “TRACE_EXPLORATION” provided by ASAP

Sweep-line Exploration Interfaces

- The "PROGRESS_MEASSURE" signature is defined by the sweep-line plug-in (and only applicable fo
- The "SWEE
signature is
in, but exte
provided by ASA

Or, reiterating an earlier point:
The sweep-line method
depends on previously defined
interfaces and implements one
of these interfaces

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitialStates : unit -> (state * event list) list

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string representation of states and events --- *)
    val stateToString : state -> string
        ToString : event -> string
```

A Couple Interfaces: MODEL

Abstraction of states and events

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitialStates : unit -> (state * event list) list

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string representation of states and events --- *)
    val stateToString : state -> string
        ToString : event -> string
```

A Couple Interfaces: MODEL

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitialStates : unit -> (state * event list) list

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string representation of states and events --- *)
    val stateToString : state -> string
    val eventToString : event -> string
```

Abstraction of states and events

The the initial state(s) of the model

A Couple Interfaces: MODEL

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitialStates : unit -> (state * event list) list

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string representation of states and events --- *)
    val stateToString : state -> string
        ToString : event -> string
```

Abstraction of states and events

The the initial state(s) of the model

...and enabled events in each state

A Couple Interfaces: MODEL

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitial

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string representation of states and events --- *)
    val stateToString : state -> string
    val eventToString : event -> string
```

Abstraction of states and events

The the initial state(s) of the model

...and enabled events in each state

Successors from executing an event in a state

A Couple Interfaces: MODEL

```
signature MODEL =
sig

    eqtype state
    eqtype event

    exception EventNotEnabled

    (* --- get the initial states and enabled events in each state --- *)
    val getInitial

    (* --- get the successor states and enabled events in each successor state --- *)
    val nextStates : state * event -> (state * event list) list

    (* --- execute an event sequence and return the list of resulting states and enabled events --- *)
    val executeSequence : state * event list -> (state * event list) list

    (* --- string repr
    val stateToString
        ToString : event -> string
```

Abstraction of states and events

The the initial state(s) of the model

...and enabled events in each state

Successors from executing an event in a state

...or a sequence of events

A Couple Interfaces: MODEL

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

Functions that allow us to gather information about states and events (like fold for lists in SML)

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state *
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

Functions that allow us to gather information about states and events (like fold for lists in SML)

A (empty or non-empty) storage for "permanent" storage of states (V)

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state *
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

Functions that allow us to gather information about states and events (like fold for lists in SML)

A (empty or non-empty) storage for "permanent" storage of states (V)

A (list of) state(s) from which to start exploration

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state *
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

Functions that allow us to gather information about states and events (like fold for lists in SML)

A (empty or non-empty) storage for "permanent" storage of states (V)

A (list of) state(s) from which to start exploration

The storage after exploration

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state *
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

Functions that allow us to gather information about states and events (like fold for lists in SML)

A (empty or non-empty) storage for "permanent" storage of states (V)

A (list of) state(s) from which to start exploration

The storage after exploration

Values computed by state_hook and arc_hook

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

A Couple Interfaces: EXPLORATION

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

```
fun state_hook property ((state, events), errors) =
    if property (state, events)
    then errors
    else state::errors
```

```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

```
fun state_hook property ((state, events), errors) =
    if property (state, events)
    then errors
    else state::errors
```

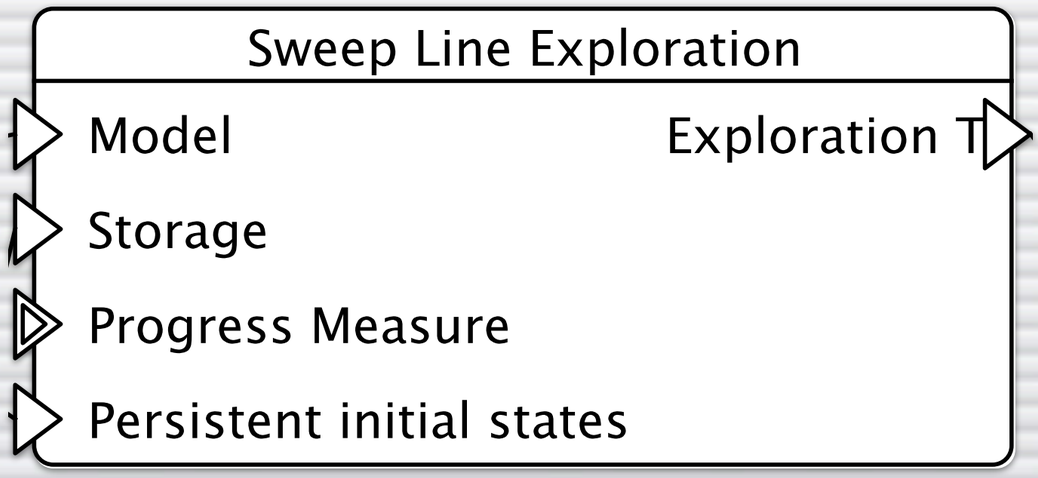
```
signature SIMPLE_EXPLORATION =
sig
    eqtype state
    eqtype event
    type 'a storage

    val explore :
    {a_initial: 'a,
    arc_hook: ((state * event list) * event * (state * event list)) * 'a -> 'a,
    s_initial: 'b,
    state_hook: (state * event list) * 'b -> 'b
    }
    ->
    'c storage
    ->
    (state * event list) list
    -> 'c storage * 'b * 'a
end
```

```
fun state_hook property ((state, events), errors) =
    if property (state, events)
    then errors
    else state::errors
```

Build a list of states violating “property”

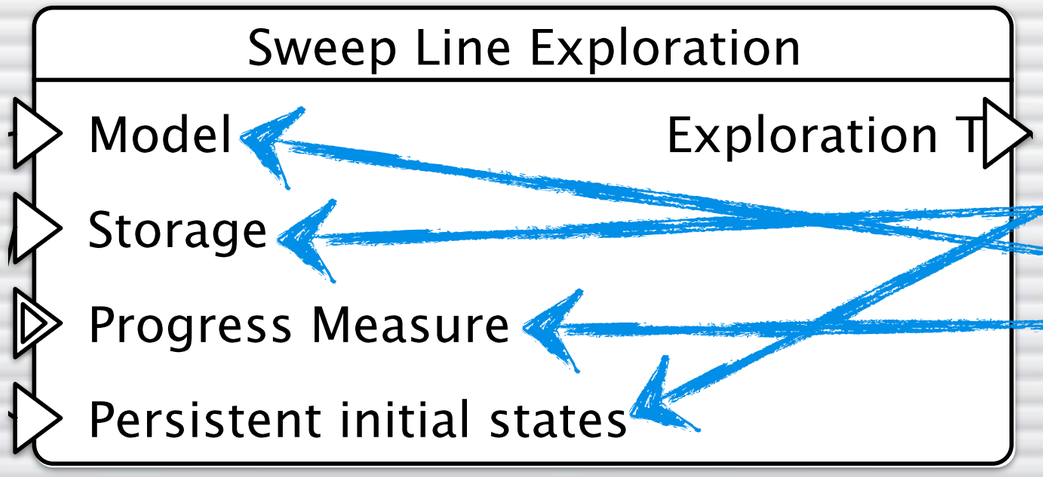
Extending JoSEL

- When we have developed a new method, we wish to integrate it into the GUI of ASAP
- JoSEL can be extended by adding new tasks (ASAP defines an extension point for this)
- We basically need to create a task for each functor we create
- EMF makes all the boiler-plate code for us, and ASAP contains abstract classes that do most of the work



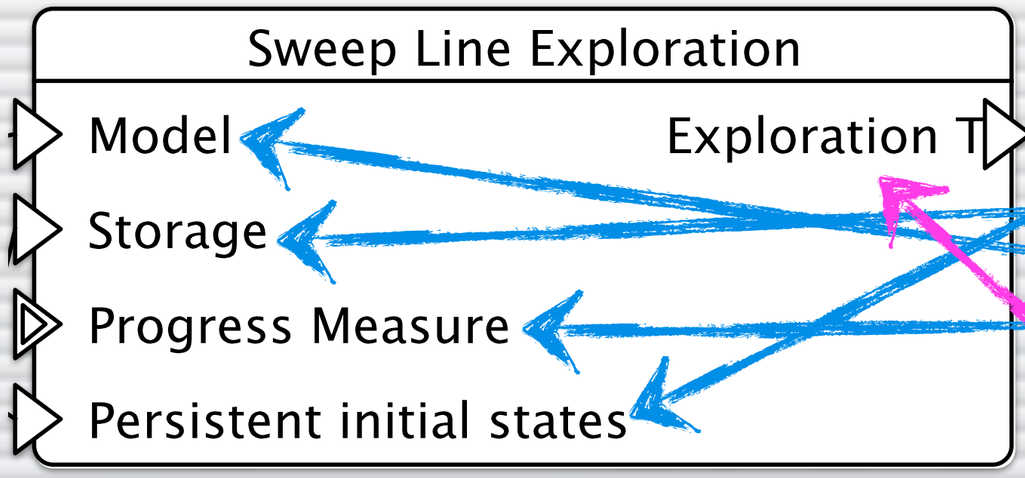

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Example: Sweep-line Exploration




```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

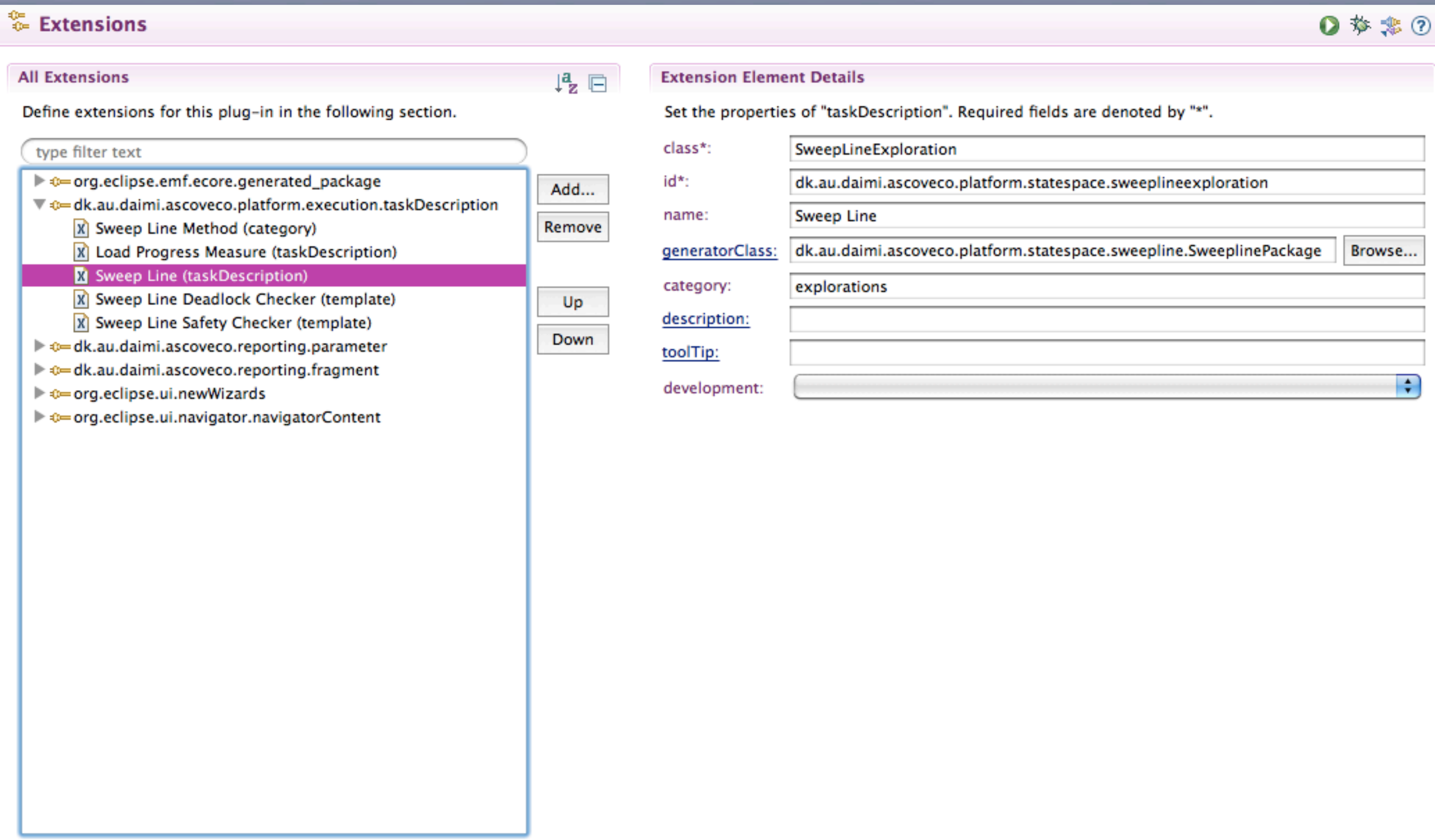
Example: Sweep-line Exploration



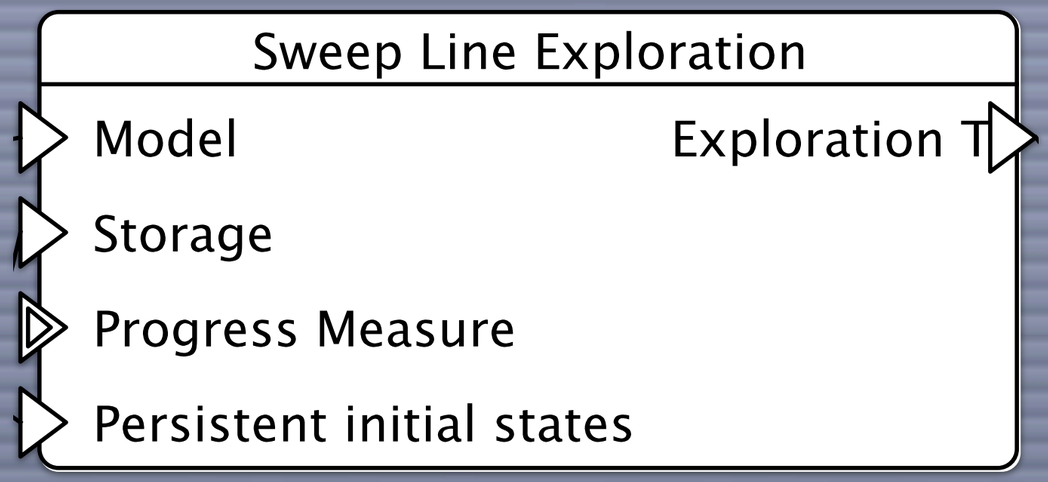

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Example: Sweep-line Exploration

Sweep-line Task

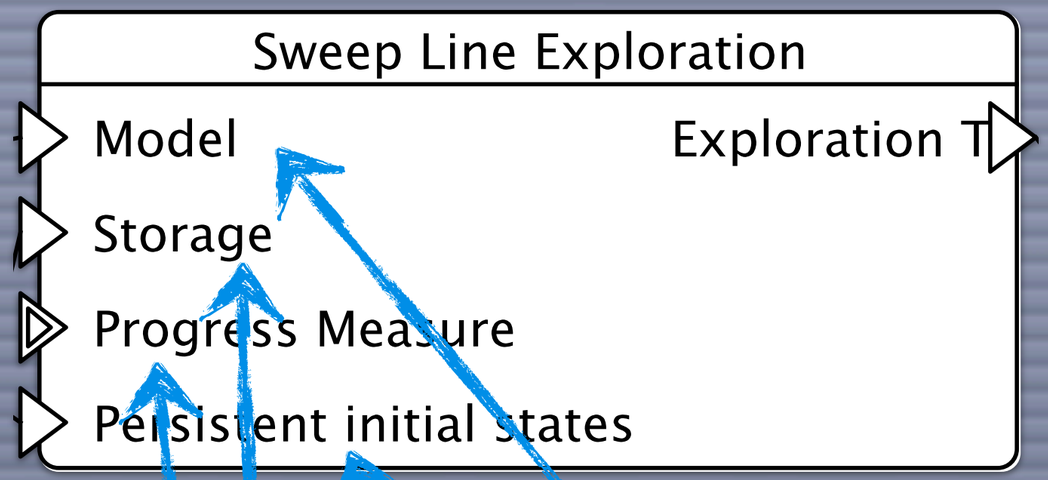


Tasks



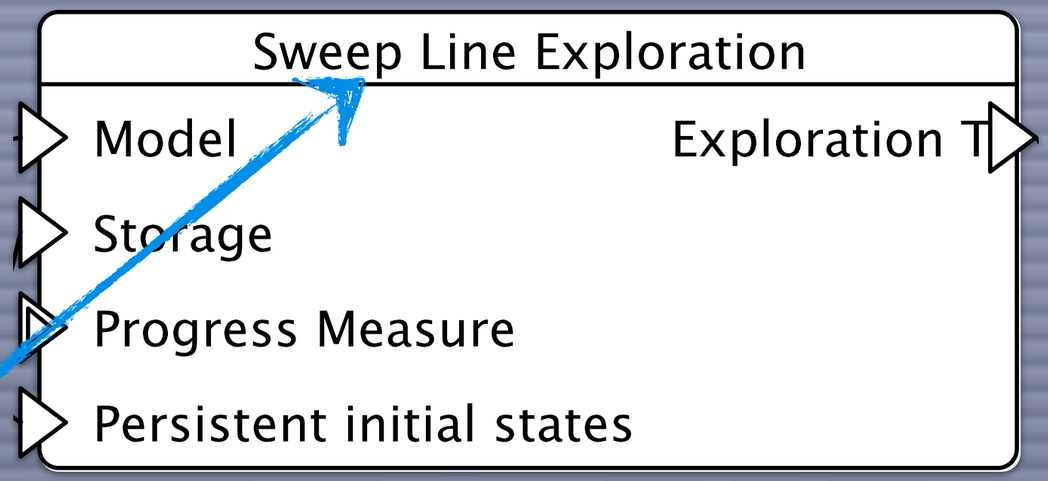

```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

Tasks




```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

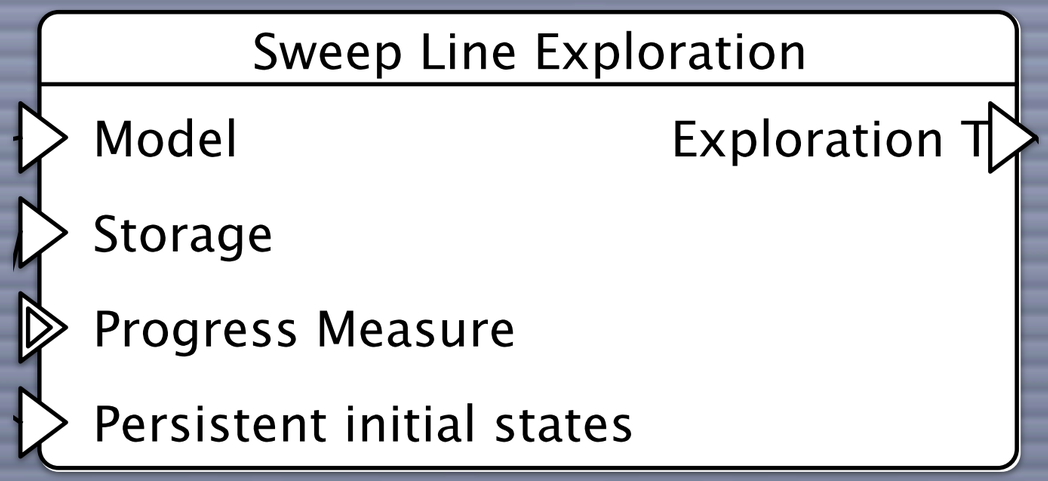
Tasks




```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

Tasks

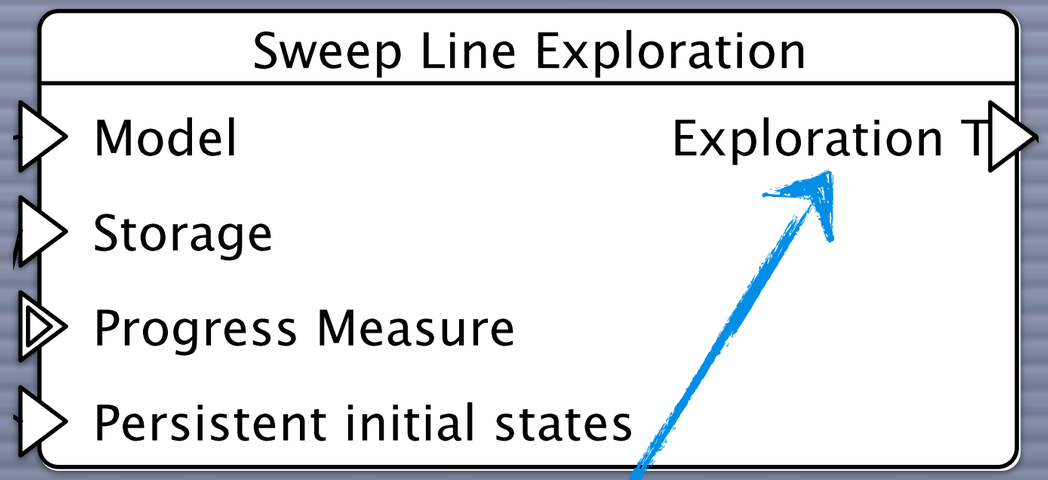



```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

Tasks



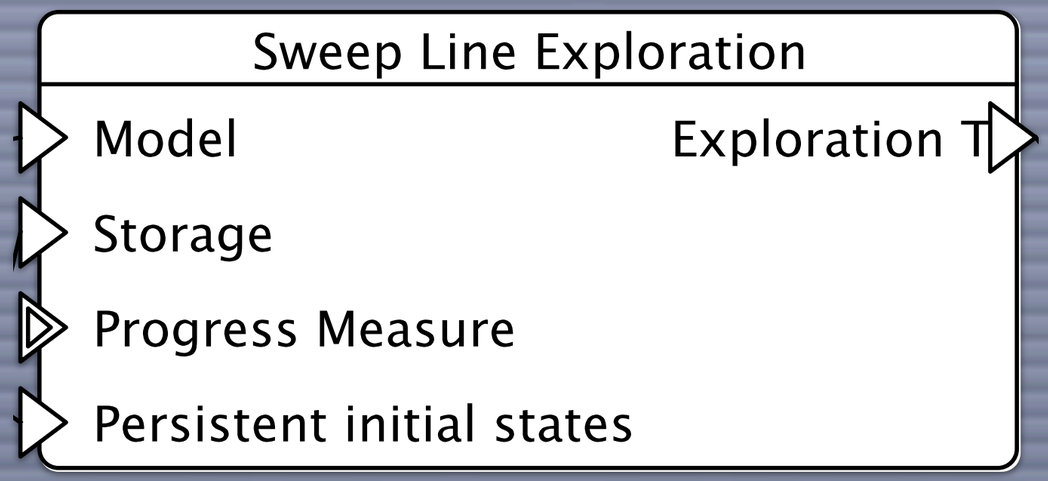

```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

```
public ValueDescription getReturnType() {
    return ExecutionFactory.eINSTANCE.createValueDescription("Exploration T", MLTraceExploration.class);
}
```

Tasks




```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

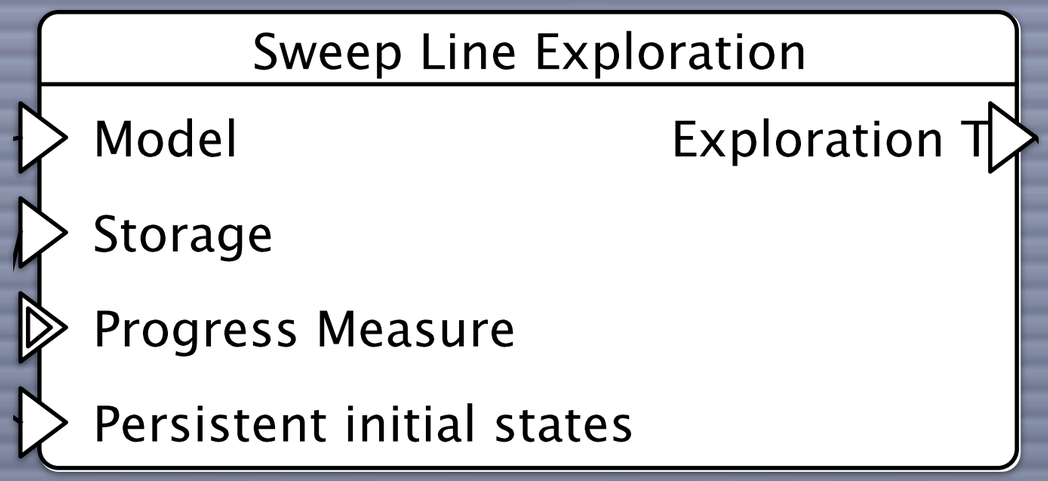
```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

```
public ValueDescription getReturnType() {
    return ExecutionFactory.eINSTANCE.createValueDescription("Exploration T", MLTraceExploration.class);
}
```

```
public <V, E> MLTraceExploration executeTask(final MLModel<V, E> model, final MLRemoveStorage storage,
        final MLProgressMeasure<V, E> progressMeasure, final Boolean setInitAsPersistent) throws Exception {
    final MLTraceExploration result = new MLTraceExploration(model.getSimulator(), null);
    model.getSimulator().evaluate(
            result.getDeclaration() + " = IntermediateStatsExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = StoppableExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = " + getFunctorName()
            + "(structure Model = " + model.getStructureName()
            + " structure Storage = " + storage.getStructureName()
            + " structure Measure = " + progressMeasure.getStructureName()
            + " val markInitStatesAsPersistent = " + setInitAsPersistent + ")))");
```

Tasks




```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

```
public <V, E> MLTraceExploration executeTask(final MLModel<V, E> model, final MLRemoveStorage storage,
        final MLProgressMeasure<V, E> progressMeasure, final Boolean setInitAsPersistent) throws Exception {
    final MLTraceExploration result = new MLTraceExploration(model.getSimulator(), null);
    model.getSimulator().evaluate(
            result.getDeclaration() + " = IntermediateStatsExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = StoppableExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = " + getFunctorName()
            + "(structure Model = " + model.getStructureName()
            + " structure Storage = " + storage.getStructureName()
            + " structure Measure = " + progressMeasure.getStructureName()
            + " val markInitStatesAsPersistent = " + setInitAsPersistent + "))");
    result.addChild(model, storage, progressMeasure);
    return result;
}
```

New Template

Extensions

All Extensions

Define extensions for this plug-in in the following section.

type filter text

- org.eclipse.emf.ecore.generated_package
- dk.au.daimi.ascoveco.platform.execution.taskDescription
  - Sweep Line Method (category)
  - Load Progress Measure (taskDescription)
  - Sweep Line (taskDescription)
  - Sweep Line Deadlock Checker (template)
  - Sweep Line Safety Checker (template)
- dk.au.daimi.ascoveco.reporting.parameter
- dk.au.daimi.ascoveco.reporting.fragment
- org.eclipse.ui.newWizards
- org.eclipse.ui.navigator.navigatorContent

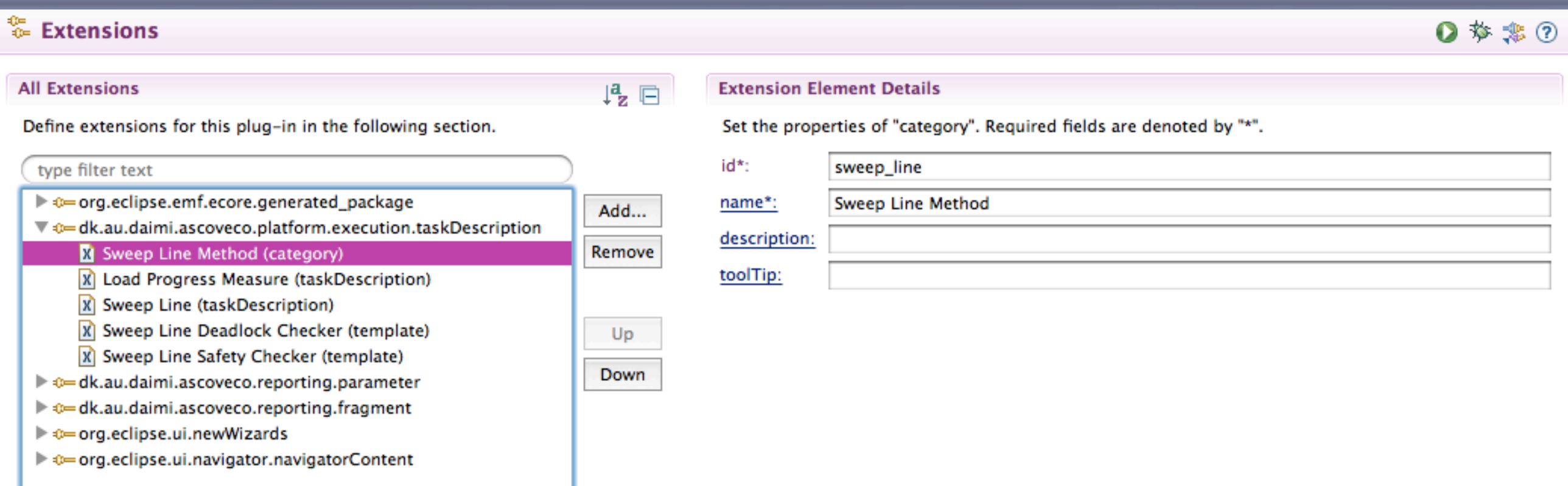
Add... | Remove | Up | Down

Extension Element Details

Set the properties of "template". Required fields are denoted by "*".

id*:	dk.au.daimi.ascoveco.platform.statespace.sweepline.sweeplinesafetychecker
factory*:	dk.au.daimi.ascoveco.platform.statespace.sweepline.macros.SweepLineSaf (Browse...)
name:	Sweep Line Safety Checker
category:	sweep_line
showIn:	NEW_WIZARD
description:	Verify a safety property using the sweep line method.
toolTip:	
development:	

New Category (Palette)

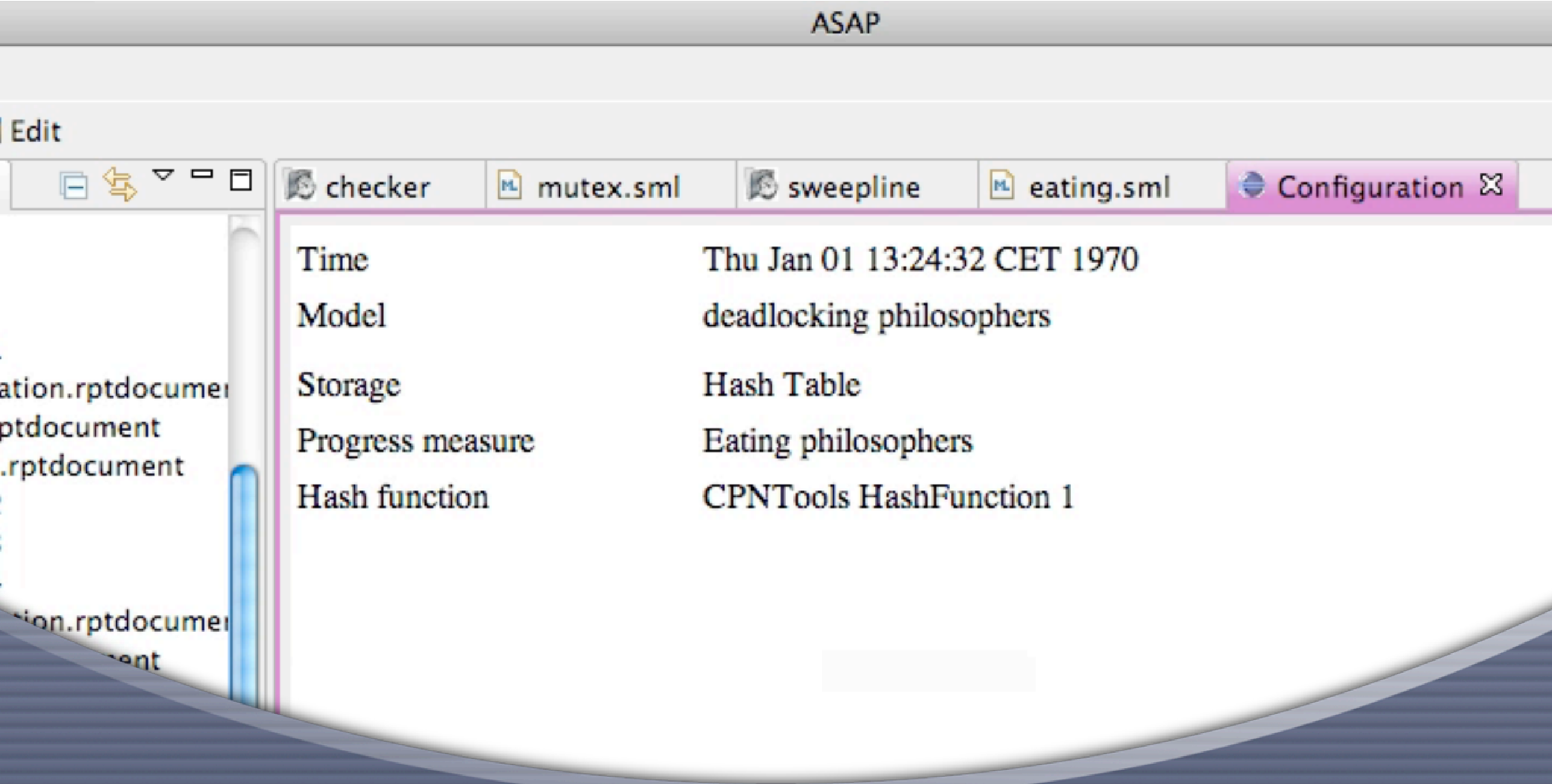


Reporting in ASAP

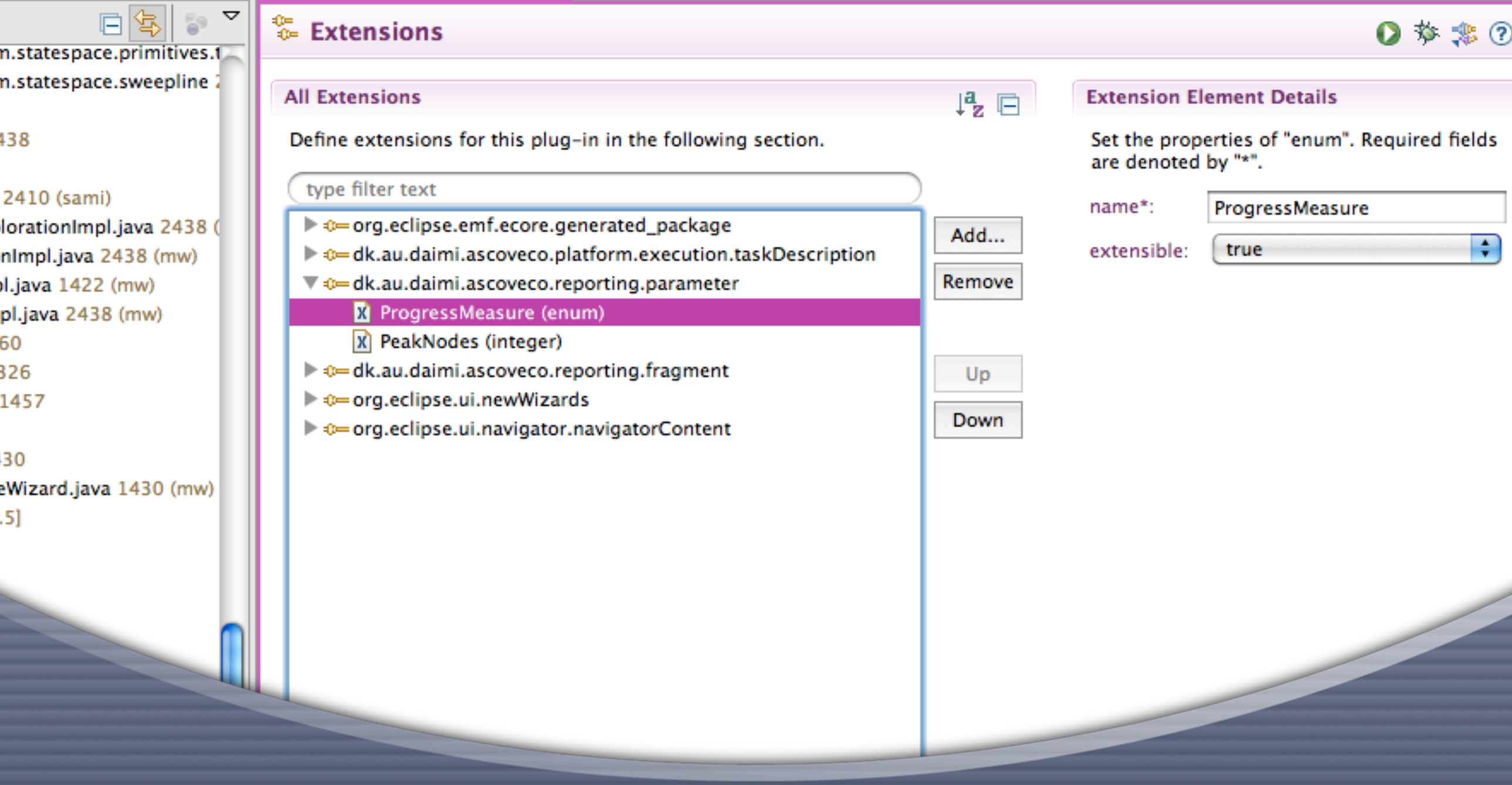
- ASAP automatically gathers information about every execution in a database (either an in-memory database or MySQL)
- The standard report is created using a standard report generating tool (BIRT)
  - ASAP is able to automatically assemble a report based on report fragments

Adding New Entries to the Report

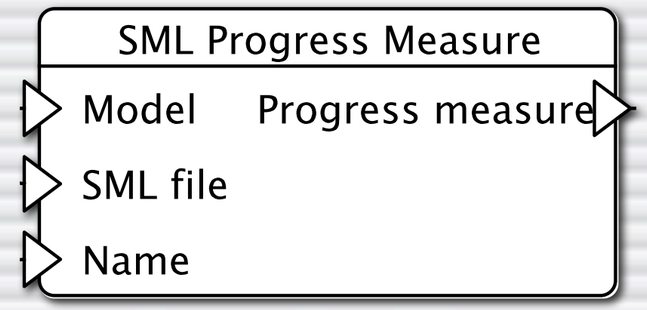
- Add a new value entry to the database
- Make sure the value is gathered during execution
- (Make a new report item model and report item presentation)
- Make a fragment showing your value



Example:
Adding PM to the Report

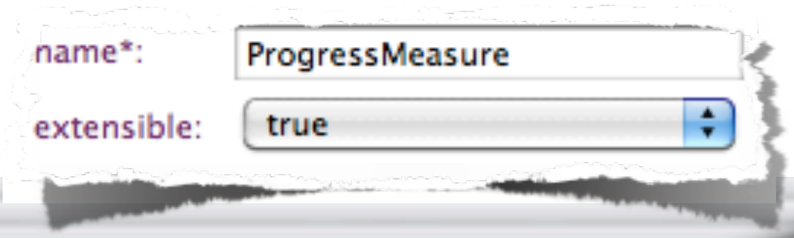


Step 1:
Add Value to Database




```
public <V, E> MLProgressMeasure<V, E> executeTask(final MLModel<V, E> model,
        final IFile pmFile, final String name) throws Exception {
    MLProgressMeasure<V, E> result = new MLProgressMeasure<V, E>(model.getSimulator(), model, name);
```

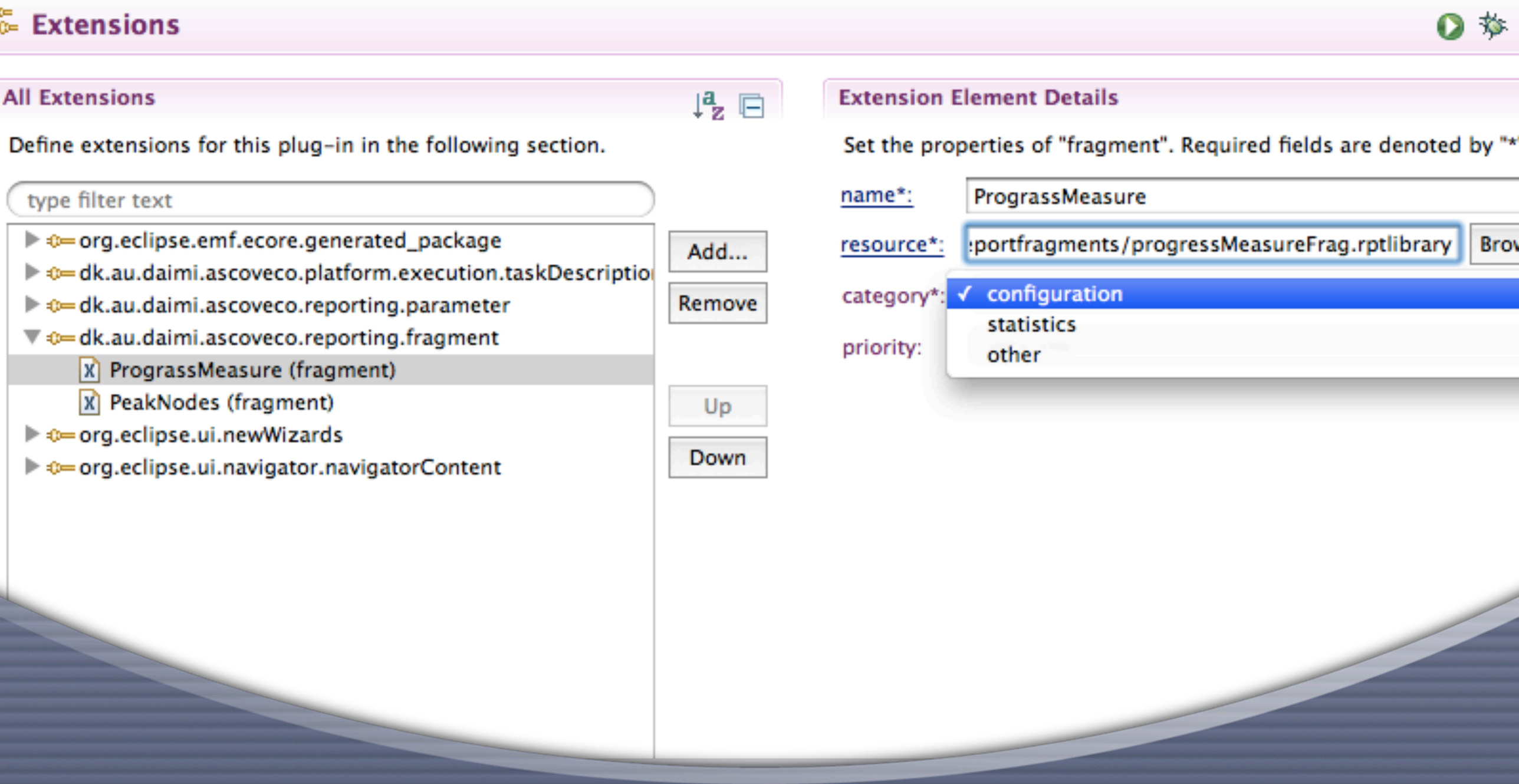
```
public MLProgressMeasure(final HighLevelSimulator simulator, final MLModel<V, E> model, final String name) {
    super(simulator);
    values.put("ProgressMeasure", name);
    addChild(model);
}
```



Step 2:
Make Sure Value is Gathered

Demo: Step 4: Creating Report Fragment

- Start reporting plugin
- Set up reporting library
- Create reporting fragment



Registering the Fragment

Step 3: Displaying New Values

- BIRT allows us to extend it to display new values
- Simple values can automatically be displayed
  - This includes charts for series or aggregated data
- We can freely describe how to display custom values

Interesting Uses of Custom Values

- We can add the model file to the report
  - Display the model as part of the report instead of just the name
- We can (and do) add error traces or even entire state space graphs to reports
  - Currently we display dead states and error traces textually
  - It is fairly easy to display states as actual markings
  - We can show graph fragments graphically

checker | Configuration | Statistics | Results

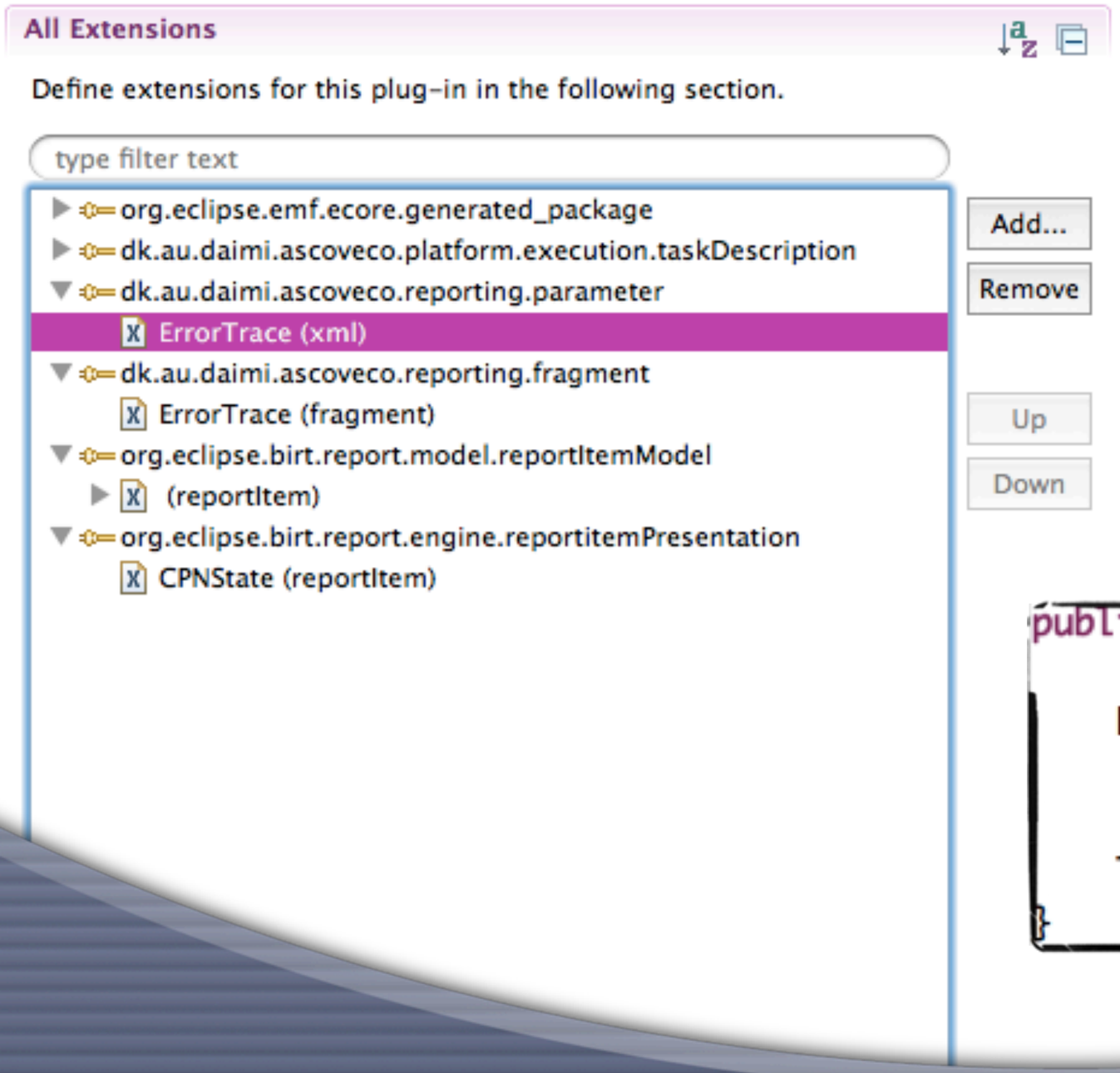
y Checker
e Report

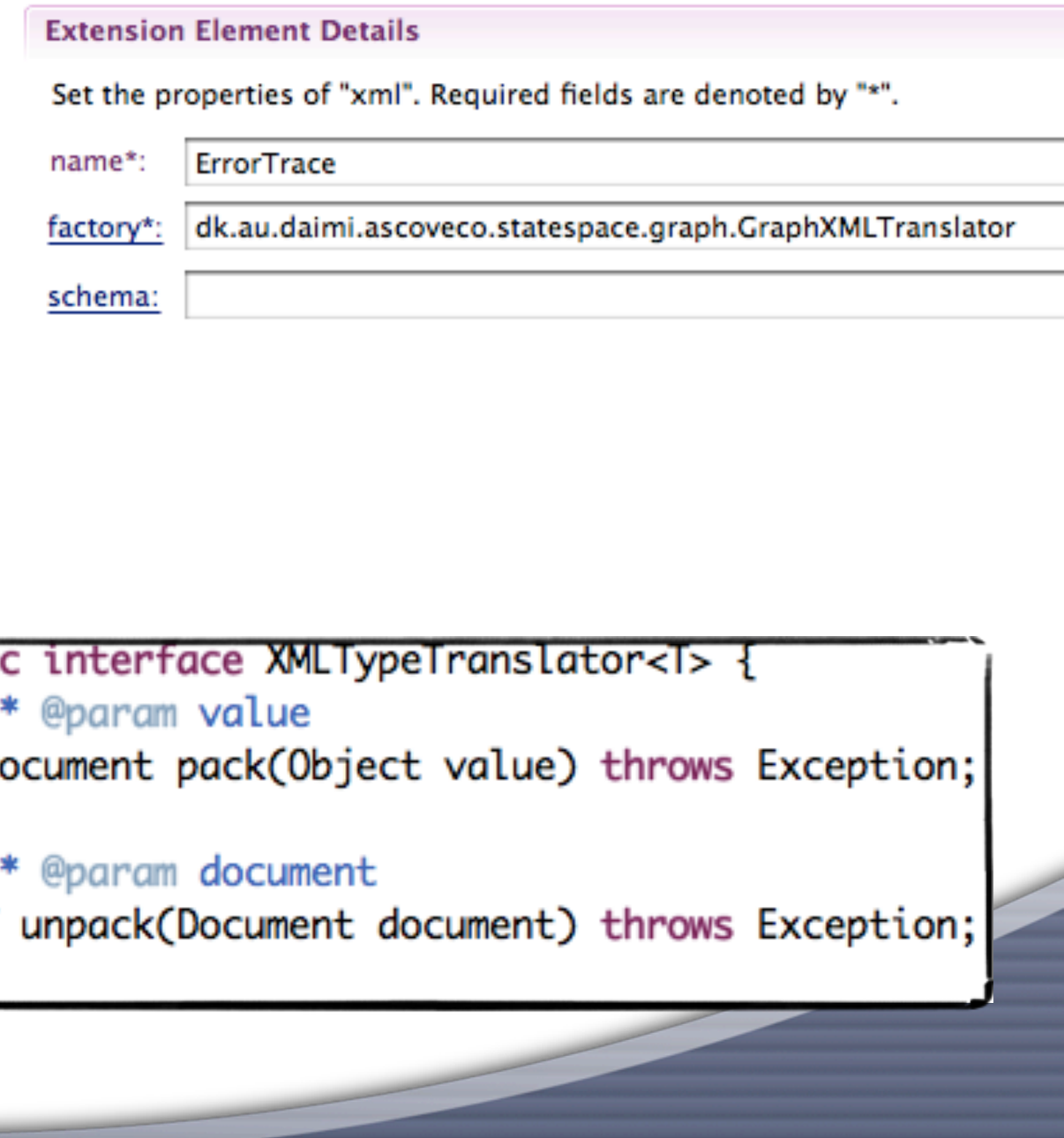
ilosophers.mod

n.rptdocument
ocument
document

No Dead States	false
Error trace	New_Page.Has_One: 1`(0,0) ++ 1`(1,1) ++ 1`(2,2) ++ 1`(3,3) ++ 1`(4,4) New_Page.Philosophers: 1`5
Error trace	New_Page.Has_One: 1`(0,1) ++ 1`(1,2) ++ 1`(2,3) ++ 1`(3,4) ++ 1`(4,0) New_Page.Philosophers: 1`5

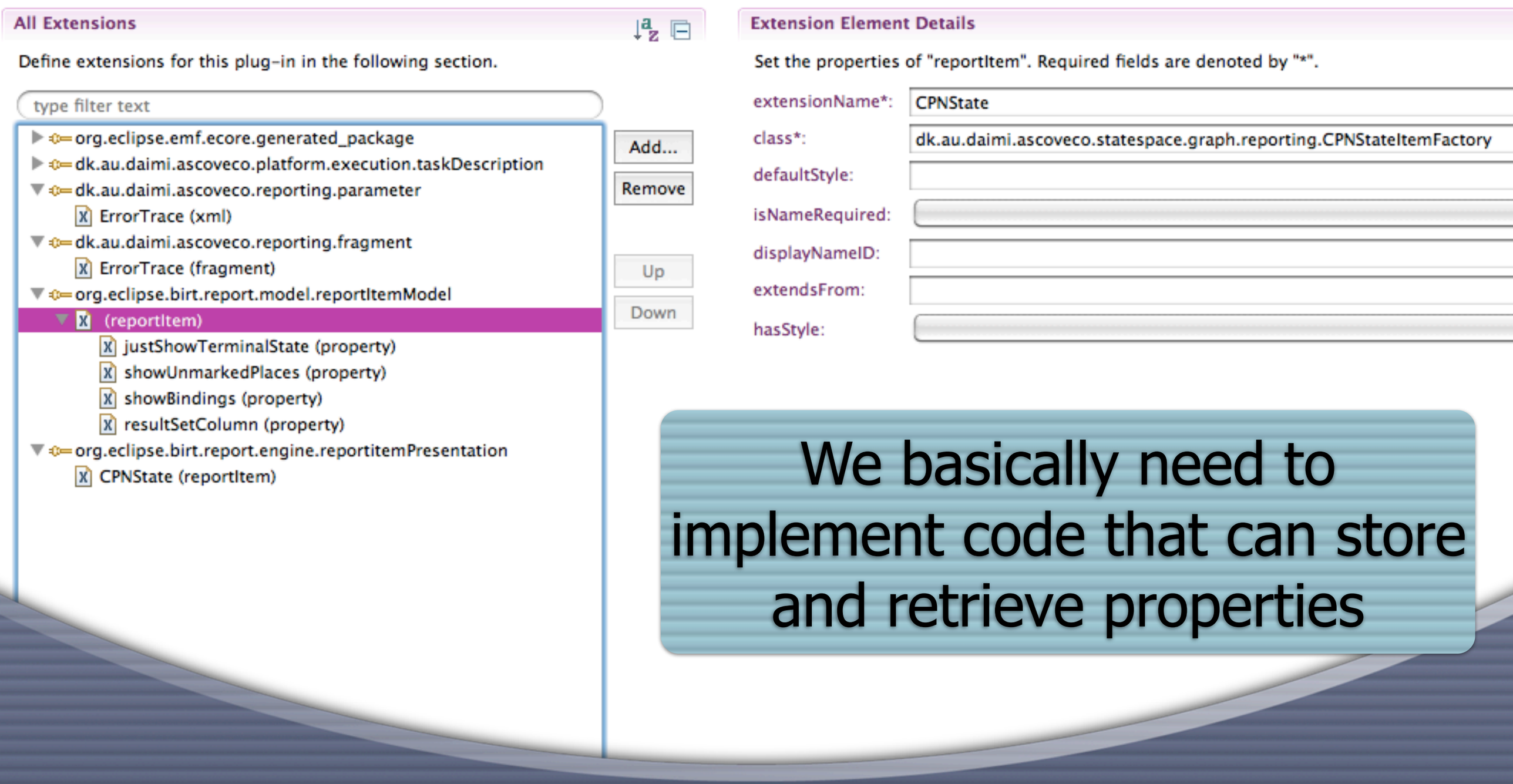
Example: Error Traces





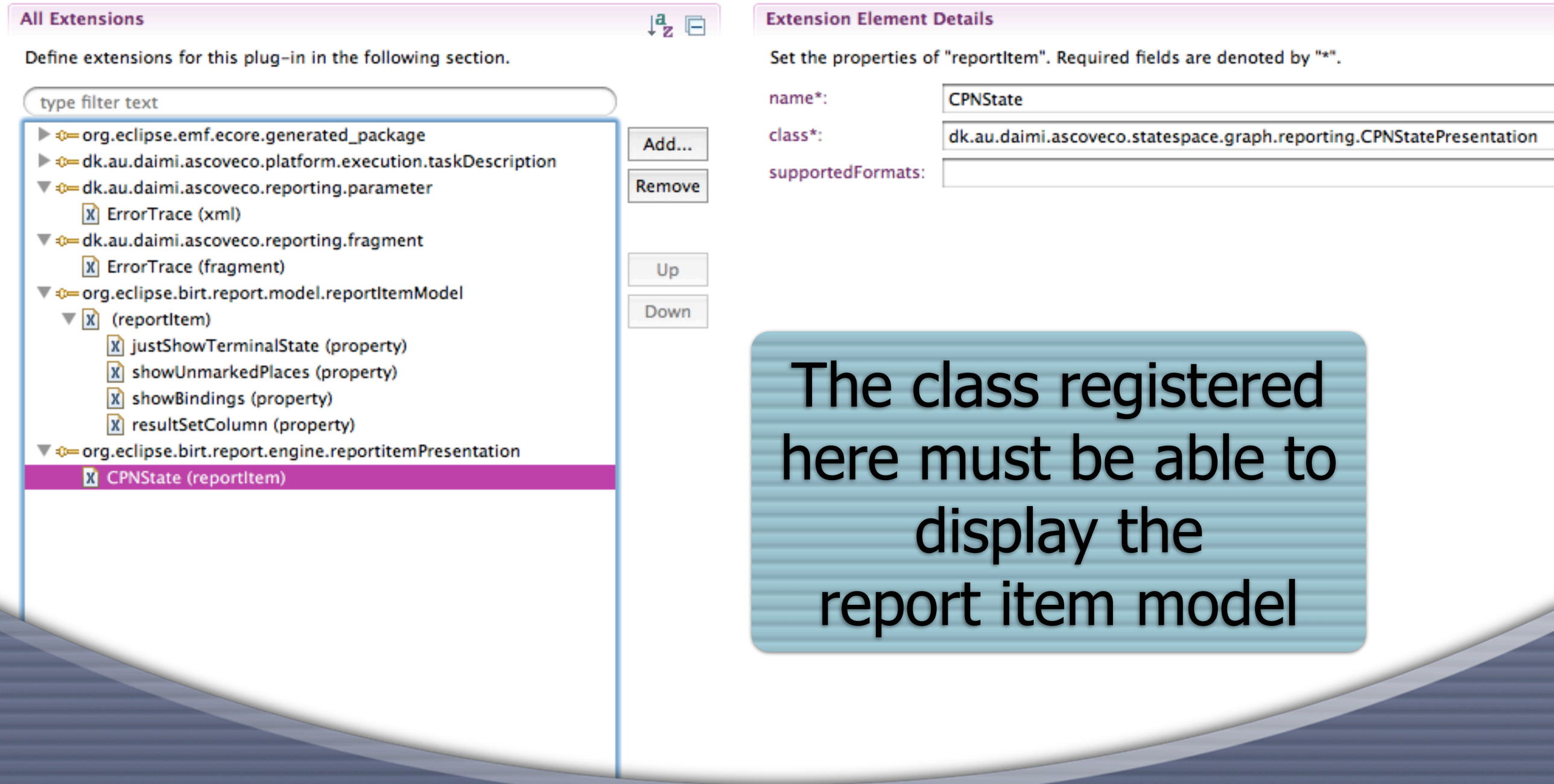

```
public interface XMLTypeTranslator<T> {
    * @param value
    Document pack(Object value) throws Exception;

    * @param document
    T unpack(Document document) throws Exception;
}
```

Registering the Type



We basically need to implement code that can store and retrieve properties

Creating a Report Item Model



Creating a Report Item Presentation

A Report Item Presentation

```
public int getOutputType() {
    return OUTPUT_AS_TEXT;
}
```

Other possibilities:
OUTPUT_AS_IMAGE
OUTPUT_AS_HTML_TEXT

A Report Item Presentation

```
public int getOutputType() {
    return OUTPUT_AS_TEXT;
}
```

```
public Object onRowSets(final IBaseResultSet[] res
    if (cpnState == null) { return null; }
    for (final IBaseResultSet resultSet : resultSe
        try {
            final SingleQueryResultSet singleQuery
            final String resultSetColumn = cpnStat
            final String xml = singleQueryResultSe
            final DirectedMultigraph<SSNode<EObjec
                new GraphXMLTranslator<EObject, EO
            final BreadthFirstIterator<SSNode<EObj
                new BreadthFirstIterator<SSNode<EO
            SSNode<EObject> node = null;
            while (iterator.hasNext()) {
                node = iterator.next();
            }
            if (node != null) { return node.getSta
        } catch (final Exception e) {
            e.printStackTrace();
            return e.toString();
        }
    }
    return "Hello world 2";
}
```

```
{
final SingleQueryResultSet singleQueryResultSet = (Sing
final String resultSetColumn = cpnState.getResultSetCol
final String xml = singleQueryResultSet.getString(resul
final DirectedMultigraph<SSNode<EObject>, SSEdge<EObje
    new GraphXMLTranslator<EObject, EObject>().unpack(x
final BreadthFirstIterator<SSNode<EObject>, SSEdge<EOb
    new BreadthFirstIterator<SSNode<EObject>, SSEdge<E
SSNode<EObject> node = null;
while (iterator.hasNext()) {
    node = iterator.next();
}
if (node != null) { return node.getState().toString();
catch (final Exception e) {
  e.printStackTrace();
  return e.toString();


"Hello world 2";
```

A Report Item Presentation

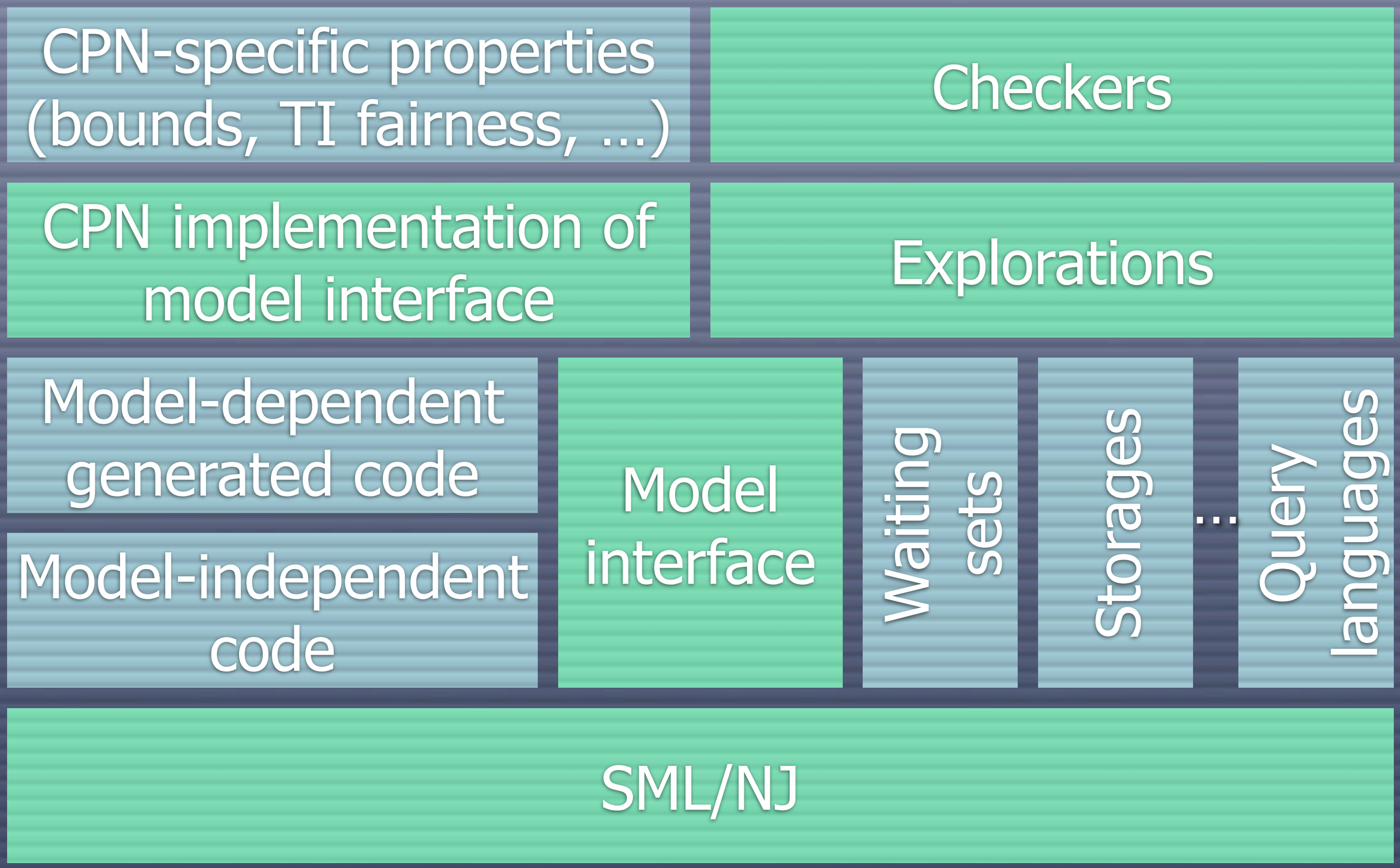
```
public int getOutputType() {
    return OUTPUT_AS_TEXT;
}
```

```
public Object onRowSets(final IBaseResultSet[] res
    if (cpnState == null) { return null; }
    for (final IBaseResultSet resultSet : resultSe
        try {
            final SingleQueryResultSet singleQuery
            final String resultSetColumn = cpnStat
            final String xml = singleQueryResultSe
            final DirectedMultigraph<SSNode<EObjec
                new GraphXMLTranslator<EObject, EO
            final BreadthFirstIterator<SSNode<EObj
                new BreadthFirstIterator<SSNode<EO
            SSNode<EObject> node = null;
            while (iterator.hasNext()) {
                node = iterator.next();
            }
            if (node != null) { return node.getSta
        } catch (final Exception e) {
            e.printStackTrace();
            return e.toString();
        }
    }
    return "Hello world 2";
}
```

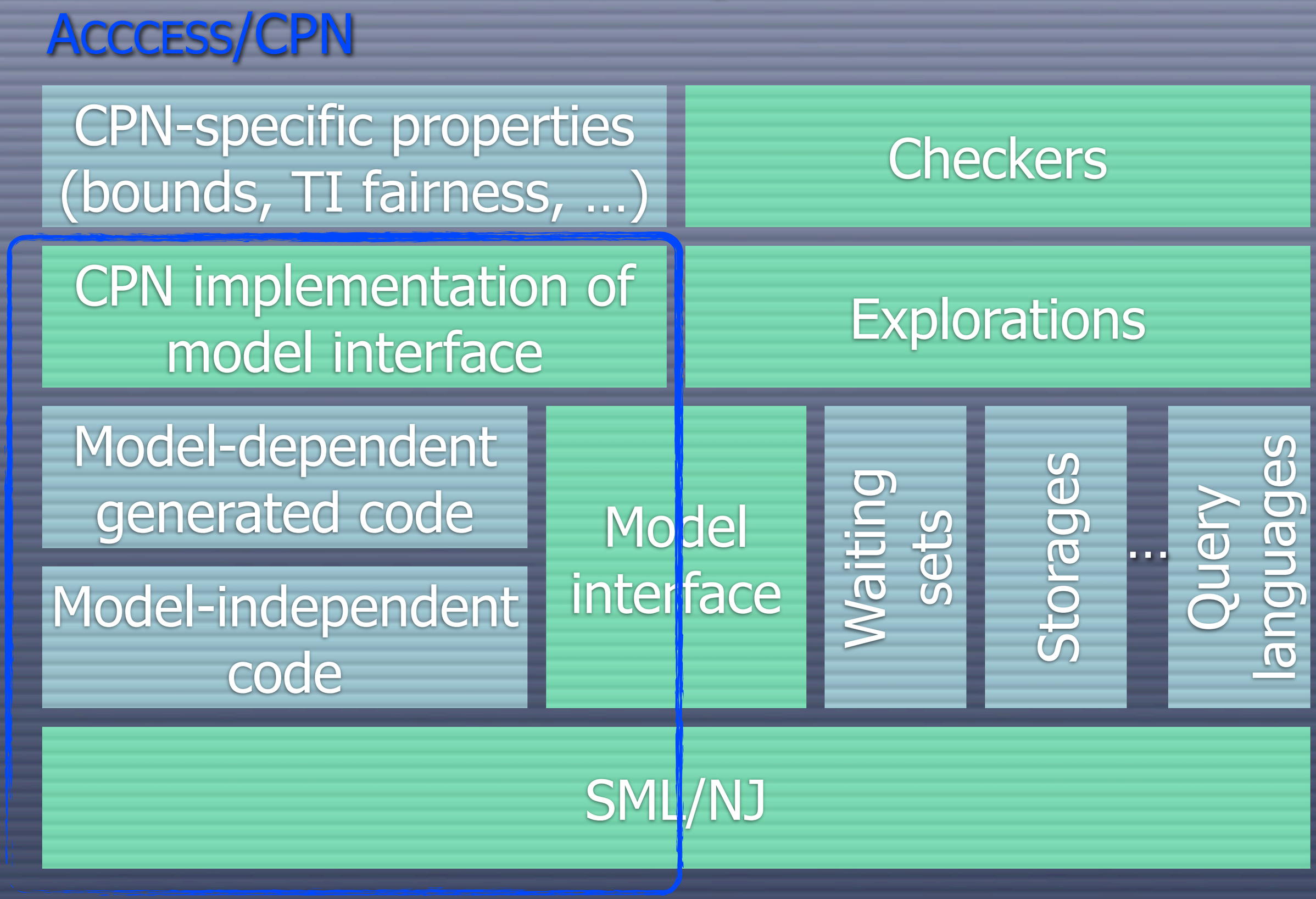
Access/CPN

- We have isolated the library used by ASAP to load CPN models as well as the interface used by the state space engine
- These two parts together are distributed under the name Access/CPN

ACCESS/CPN

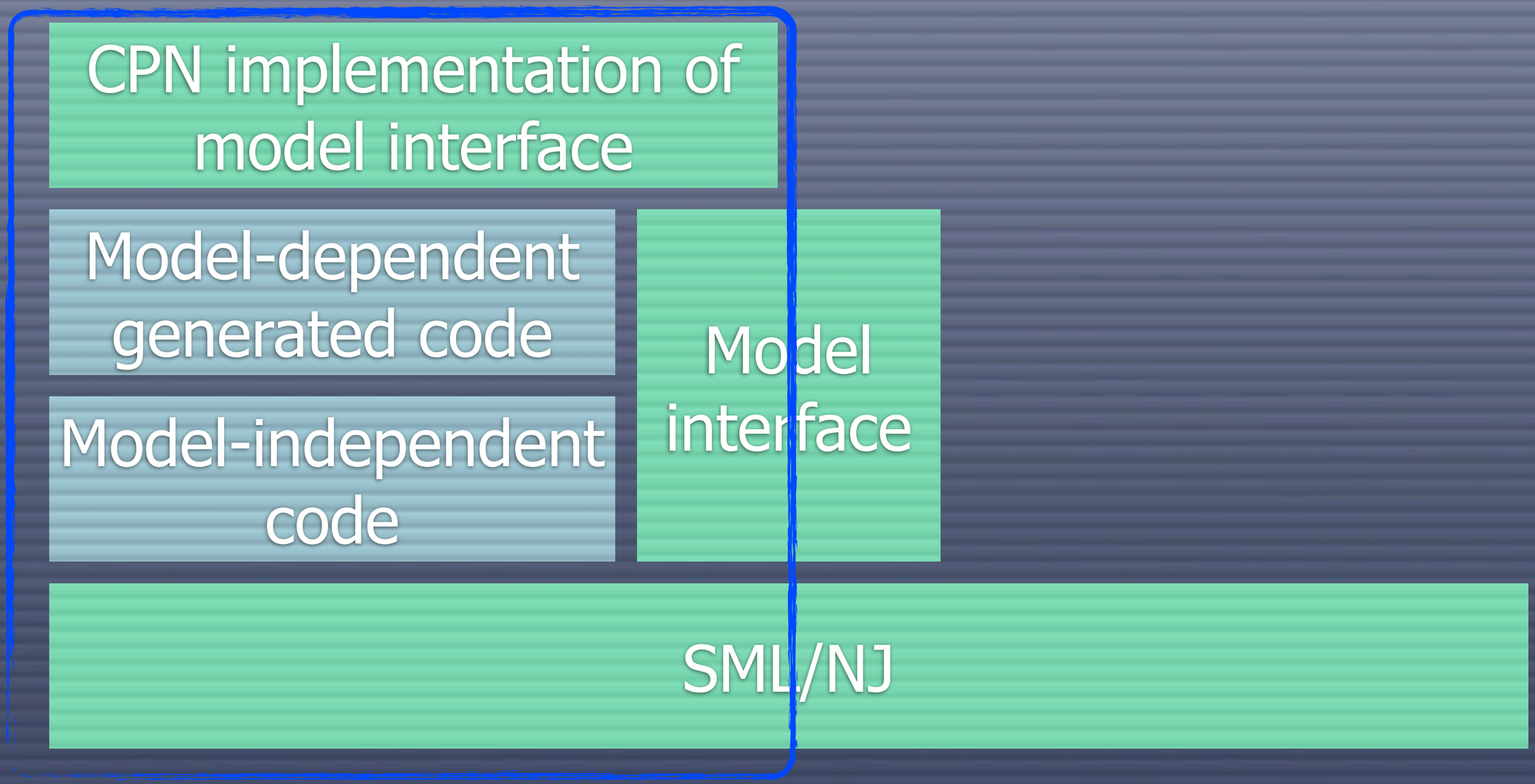

CPN-specific properties (bounds, TI fairness, ...)
Checkers
CPN implementation of model interface
Explorations
Model-dependent generated code
Model-independent code
Model interface
Waiting sets
Storages
...
Query languages
SML/NJ

ACCESS/CPN


ACCCESS/CPN
CPN-specific properties (bounds, TI fairness, ...)
Checkers
CPN implementation of model interface
Explorations
Model-dependent generated code
Model-independent code
Model interface
Waiting sets
Storages
...
Query languages
SML/NJ

Access/CPN

Acccess/CPN



Access/CPN Features

- With Access/CPN you can:
  - Load models from CPN Tools
  - Simulate models programmatically (both automatic and "manual")
  - Inspect and change state
  - Evaluate SML code
  - Build a state space tool :-)

Access/CPN Uses

- ASAP
- Cosimulation of SystemC and CP-nets
- Code-generation from CPN models
- Integration into ProM (R. Mans & M. Netjes)
- …