zeligsoft

Validating Eclipse Models

How can I ensure valid models?

zeligsoft

Overview

- Explore the validation service
- Explore the different types of constraints

Toby McClean, Zeligsoft, 2008

zeligsoft

Goals

- After this module you understand how validation is performed in the Eclipse environment
- You will be able to add your own constraints

zeligsoft

Agenda

- Overview
- Static and dynamic constraint providers
- Constraints
- Validation service
- Creating constraints

zeligsoft

What can I Validate?

- The validation framework provides a way to describe constraints to go with models
  - To protect model sanity
  - To validate domain rules
  - To validate your specific rules

zeligsoft

Validation Framework Overview

- Constraints are organized through categories and bindings
- Constraint providers contribute constraints
- Constraint parsers implement implementation languages
- Traversal strategies walk the model
- Validation listeners are notified when validation occurs
- Notification generators for custom (esp. higher-order) notifications for processing in live validation

zeligsoft

What does the Validation Framework Provide?

- Invocation (Triggers)
  - "Batch" validation: initiated by the user or by the system on some important event, validates all or a selected subset of a model
  - "Live" validation: initiated automatically by the system to validate a set of changes performed during some transaction. The semantics of "transaction" are defined by the client
    - Constraints can specify which particular changes trigger them (by feature and event type)
- Support for OCL constraints
  - EMFT Validation uses the OCL component of the MDT project to provide out-of-box support for specifying constraints using OCL
  - API supports UML binding and more flexibility in working with OCL constraints embedded in metamodels

zeligsoft

Constraint Providers

- Constraint providers are of two flavours: static and dynamic
  - Static providers declare their constraints in the plugin.xml of a client plug-in of the constrained model
  - Dynamic providers obtain constraints from an arbitrary source at run-time
- Both kinds of providers can declare constraint categories and include their constraints in categories defined by any provider
  - Categories are hierarchical namespaces for constraints
  - Constraints are grouped by category in the preference page

zeligsoft

Static Constraint Provider

```
<extension point="org.eclipse.emf.validation.constraintProviders">
  <category name="Library Constraints" id="com.example.library">

  <constraintProvider>
    <package namespaceUri="http:///www.eclipse.org/Library/1.0.0"/>
    <constraints categories="com.example.library">
      <constraint
          lang="Java"
          class="com.example.constraints.UniqueLibraryName"
          severity="WARNING"
          mode="Batch"
          name="Library Must have a Unique Name"
          id="com.example.library.LibraryNameIsUnique"
          statusCode="1">
        <description>Libraries have unique names.</description>
        <message>{0} has the same name as another library.</message>
        <target class="Library"/>
      </constraint>
    </constraints>
  </constraintProvider>
</extension>
```

zeligsoft

Dynamic Constraint Provider

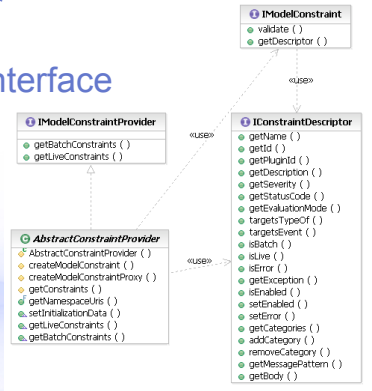
- Registered by name of a class implementing the `IModelConstraintProvider` interface
- Indicate the packages for which they supply constraints
  - Biggest difference is the absence of a `<constraints>` element
- System can optionally cache the provided constraints

```
<extension point="org.eclipse.emf.validation.constraintProviders">
  <category name="Library Constraints" id="com.example.library">

  <constraintProvider
      class="com.example.MyConstraintProvider"
      cache="false">
    <package namespaceUri="http:///www.eclipse.org/Library/1.0.0"/>
  </constraintProvider>
</extension>
```

zeligsoft

Dynamic Constraint Provider

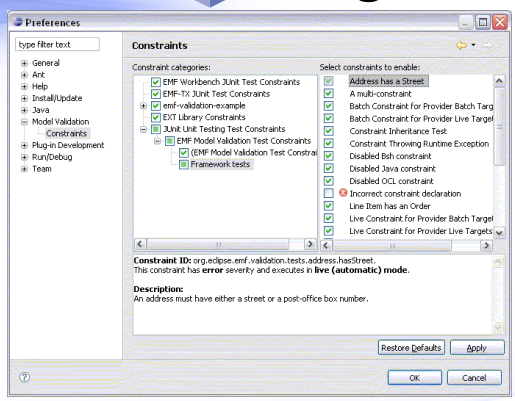
- A dynamic constraint provider implements the `IModelConstraintProvider` interface
- A descriptor supplies meta-data about a constraint
- The validation algorithm is supplied by the constraint object in the `validate()` method



zeligsoft

Constraints

- Descriptor provides:
  - ID and localizable name
  - Evaluation mode
  - Target EClasses
  - Triggering events (live)
  - Severity, error message
  - Categorization
  - Enabled state
- Much of this information appears in the preference page
- Users can choose which constraints to activate



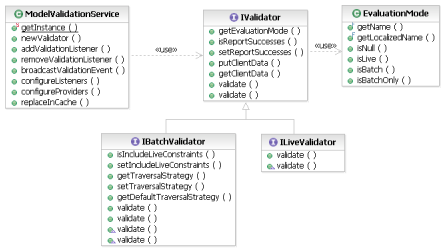
Evaluation Modes

- Batch validation is usually explicitly requested by the user, via a menu action or toolbar button
- Live validation is performed automatically as changes are made to EMF resources
  - Live constraints indicate which specific changes trigger them

```
<constraint
    mode="Live"
    … >
  <description>Libraries have unique names.</description>
  <message>{0} has the same name as: {1}.</message>
  <target class="Library">
     <event name="Set">
        <feature name="name"/>
     </event>
  </target>
</constraint>
```

Validation Service

- Evaluation of constraints is performed via the Validation Service
- The service provides validators corresponding to the available evaluation modes
- By default, batch validation includes live constraints, also, for completeness



zeligsoft

Validation Service

- To validate one or more elements, in batch mode, simply create a new validator and ask it to validate:

```
List objects = myResource.getContents(); // objects to validate

// create a validator
IValidator validator = ModelValidationService.getInstance()
      .newValidator(EvaluationMode.BATCH);

// use it!
IStatus results = validator.validate(objects);

if (!results.isOK()) {
    ErrorDialog.openError(null, "Validation", "Validation Failed",
        results);
}
```

zeligsoft

Validation Service

- Live validation does not validate objects, but rather `Notifications` indicating changes to objects

```
List<Notification> notifications = … ; // some changes that we observed

// create a validator
IValidator validator = ModelValidationService.getInstance()
      .newValidator(EvaluationMode.LIVE);

// use it!
IStatus results = validator.validate(notifications);

if (!results.isOK()) {
    ErrorDialog.openError(null, "Validation", "Validation Failed",
        results);
}
```

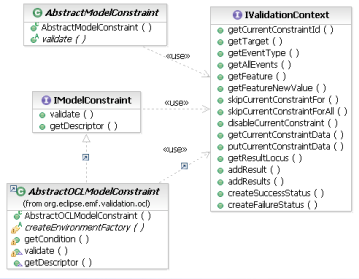
Creating Constraints

- Specifying a constraint doesn't necessarily require any code
- Requires the OCL component of the MDT project

```
<constraint
    lang="OCL"
    mode="Batch"
    … >
  <description>Libraries have unique names.</description>
  <message>{0} has the same name as: {1}.</message>
  <target class="Library"/>

  <![CDATA[
  Library.allInstances()->forAll(l |
       l <> self implies l.name <> self.name)
  ]]>

</constraint>
```

Creating Constraints

- Sometimes the easiest or best way to formulate a constraint is in Java
- Java constraints extend the `AbstractModelConstraint` class
- The validation context provides the `validate()` method with information about the validation operation



zeligsoft

Validation Context

- Provides the element being validated (the target)
- Indicates the current evaluation mode (live or batch)
  - In case of live invocation, provides the changed feature and the event type
- Allows constraints to cache arbitrary data for the duration of the current validation operation
- Provides convenient methods for reporting results
- Provides the ID of the constraint that is being invoked

IValidationContext

- getCurrentConstraintId ( )
- getTarget ( )
- getEventType ( )
- getAllEvents ( )
- getFeature ( )
- getFeatureNewValue ( )
- skipCurrentConstraintFor ( )
- skipCurrentConstraintForAll ( )
- disableCurrentConstraint ( )
- getCurrentConstraintData ( )
- putCurrentConstraintData ( )
- getResultLocus ( )
- addResult ( )
- addResults ( )
- createSuccessStatus ( )
- createFailureStatus ( )

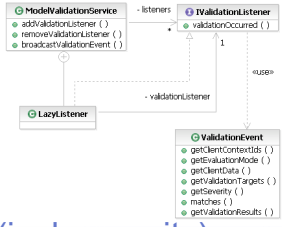
zeligsoft

Creating Constraints

```
public class LibraryNameIsUnique extends AbstractConstraint {
  public IStatus validate(IValidationContext ctx) {
    Library target = (Library) ctx.getTarget(); // object to validate

    // does this library have a unique name?
    Set<Library> libs = findLibrariesWithName(target.getName());
    if (libs.size() > 1) {
       // report this problem against all like-named libraries
       ctx.addResults(libs);

       // don't need to validate these other libraries
       libs.remove(target);
       ctx.skipCurrentConstraintFor(libs);

       return ctx.createFailureStatus(new Object[] {
          target, libs});
    }

    return ctx.createSuccessStatus();
  }
}
```

zeligsoft

Listening for Validation Events

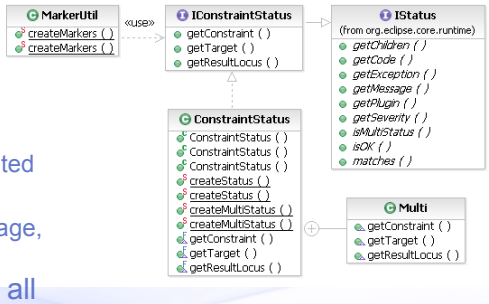


- Every validation operation executed by the Validation Service generates an event
- The event indicates the kind of validation performed, on what objects, and what were the results (incl. severity)
- Listeners registered on the extension point are lazily initialized when their selection criteria are met

```
<extension point="org.eclipse.emf.validation.validationListeners">
    <listener class="com.example.validation.ProblemsReporter">
        <clientContext id="com.example.MyClientContext"/>
    </listener>
</extension>
```

zeligsoft

Reporting Problems



- Problems are reported as IConstraintStatus objects. A status knows:
  - The constraint that was violated
  - The objects that violated it
  - The severity and error message, as usual for an IStatus
- The marker utility encodes all of this information in a problem marker on the appropriate resource, to show in the Problems view

zeligsoft

Summary

- We have discussed validation in Eclipse
- Static, dynamic, live and batch
- You have learned how to create a constraint

zeligsoft

RSA-RTE Validation

Validating models in RSA-RTE

zeligsoft

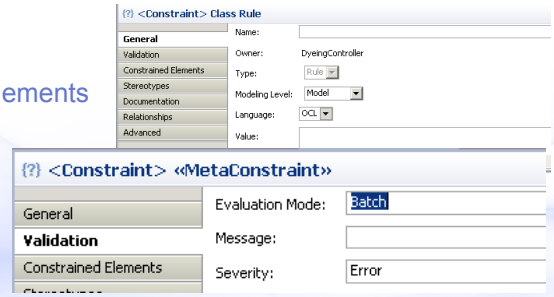
Overview

- Adding constraints to models in RSA-RTE
  - Authoring
  - Executing
- Adding constraints to profiles in RSA-RTE
  - Authoring
  - Executing

zeligsoft

Authoring Model Constraint

- Contraints can be added to model elements
  - Add UML → Constraint from the elements context menu
  - Defined by
    - Name
    - Constrained elements
    - Modeling level
    - Language
    - Mode
    - Message



zeligsoft

Executing Model Constraint

- Batch constraints are evaluated by
  - Validate in the context menu of the element
- Live constraints are evaluated by
  - Validate in the context menu
  - i.e. they are all Batch
- You will see the results if any
  - In the problems view

zeligsoft

Authoring Profile Constraint

- Profile authored in the same way as model constraints
- They apply to elements that the stereotype is applied to
- Nothing special is needed to have them work in a deployed profile
- This is specific to RSx you would have to create your own with Eclipse UML2

zeligsoft

Executing Profile Constraint

- Batch mode constraints evaluate
  - When the model or model element is validated
  - They are reported in the Problems view
- Live mode constraints evaluate
  - When the model change is made
  - They are reported in a dialog and in the Problem view

zeligsoft

Summary

- We have discussed validation specific to RSx
- How to author model specific constraints
- How to author constraints in profiles

zeligsoft

Questions