Large Models An Alternative Approach

Jos Warmer
jos.warmer@ordina.nl

Eclipse Summit 2008 Modeling Symposium
Ludwigsburg Germany

Problems with Large Models

- **Large models are problematic:**
  - multi-user access
  - version management
  - reuse of models
  - model libraries
  - long waiting times for code generation
  - Etc.

Some Solutions

- **Re positories**
  - Model stored as a graph
  - Versioning on any! model element
  - Source for code generation
- **File storage**
  - Using proxies etc.
- **Model merging**
  - Graph diffing & merging

Complex

Our Solution to Large Models

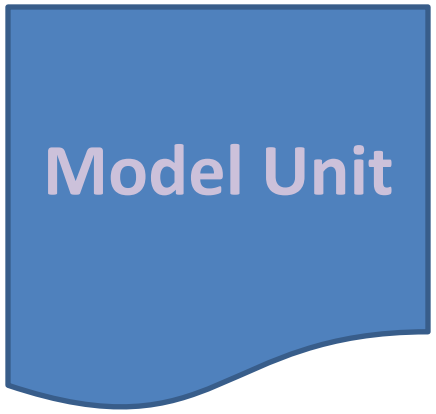
- **Don’t have large models ☺**

➔ **No problems with large models !**

Model Units

- **A Model Unit is a partial model that can be handled as a standalone entity**
  - Similar to compilation units in programming languages

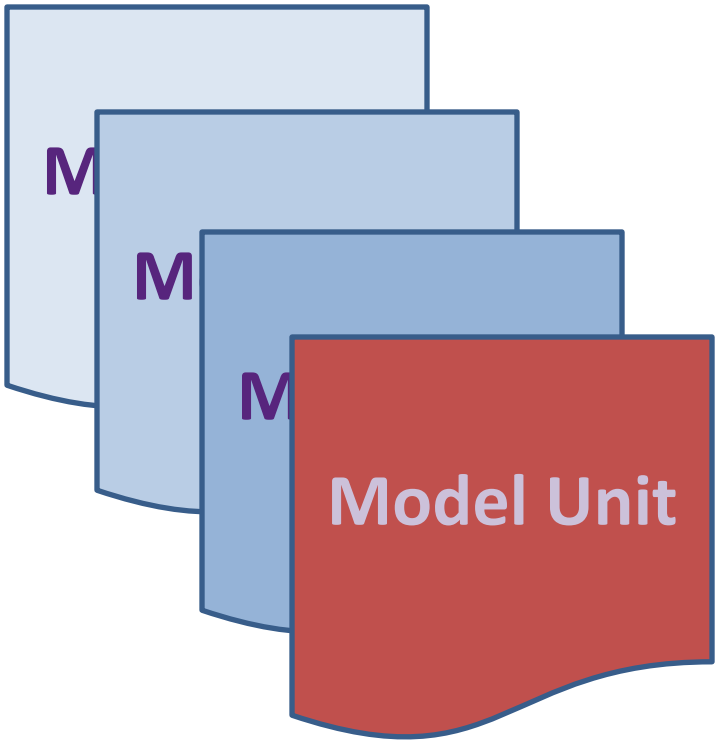
- **A Model Unit:**
  - Is stored in its own file
  - Is edited by a developer as a whole
  - Is the configuration unit
  - Is the unit of code generation



- **Model Units are part of the design of a DSL**

Model of a System

- **A Model consists of a collection of Model Units**
  - Similar to a Java program, which consists of a collection of Java Classes



TYING MODEL UNITS TOGETHER

- **A Model Unit may contain references to an element in another model unit**
  - Soft references
  - By name
    - Similar to references in a Java Class to another class

- **Reference is resolved by:**
  - Finding a model element with the right name and type in the environment
    - Similar to finding a java class anywhere on the CLASSPATH

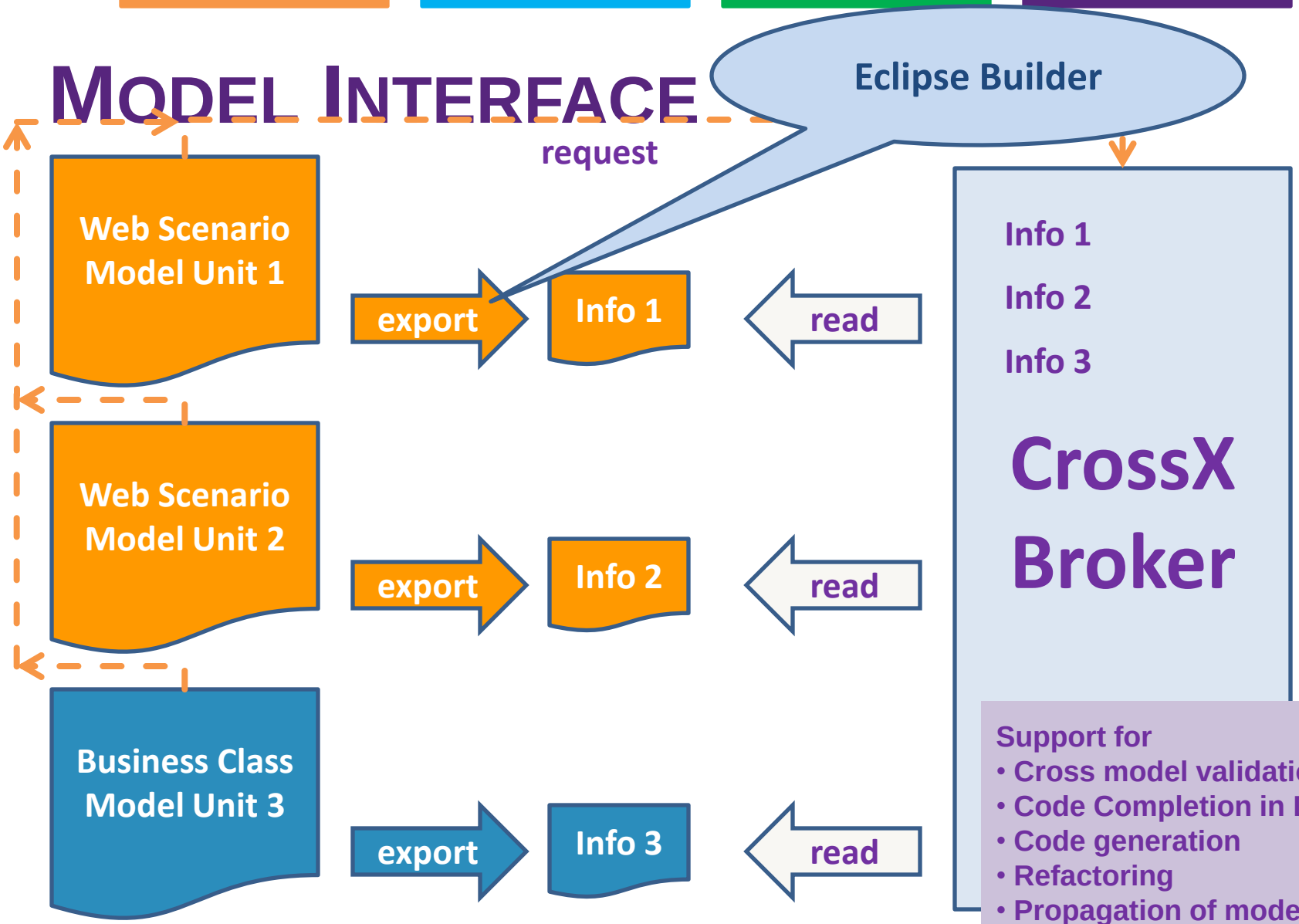
Working with Models in Eclipse

- **Check out model unit**
- **Edit model unit**
- **Save Model unit (automatic build is On)**
  - Check internal
  - Check referred model elements
  - Export model information
  - Generate code
- **Check in Model Unit**

  - This is exactly how developers work with Java programs

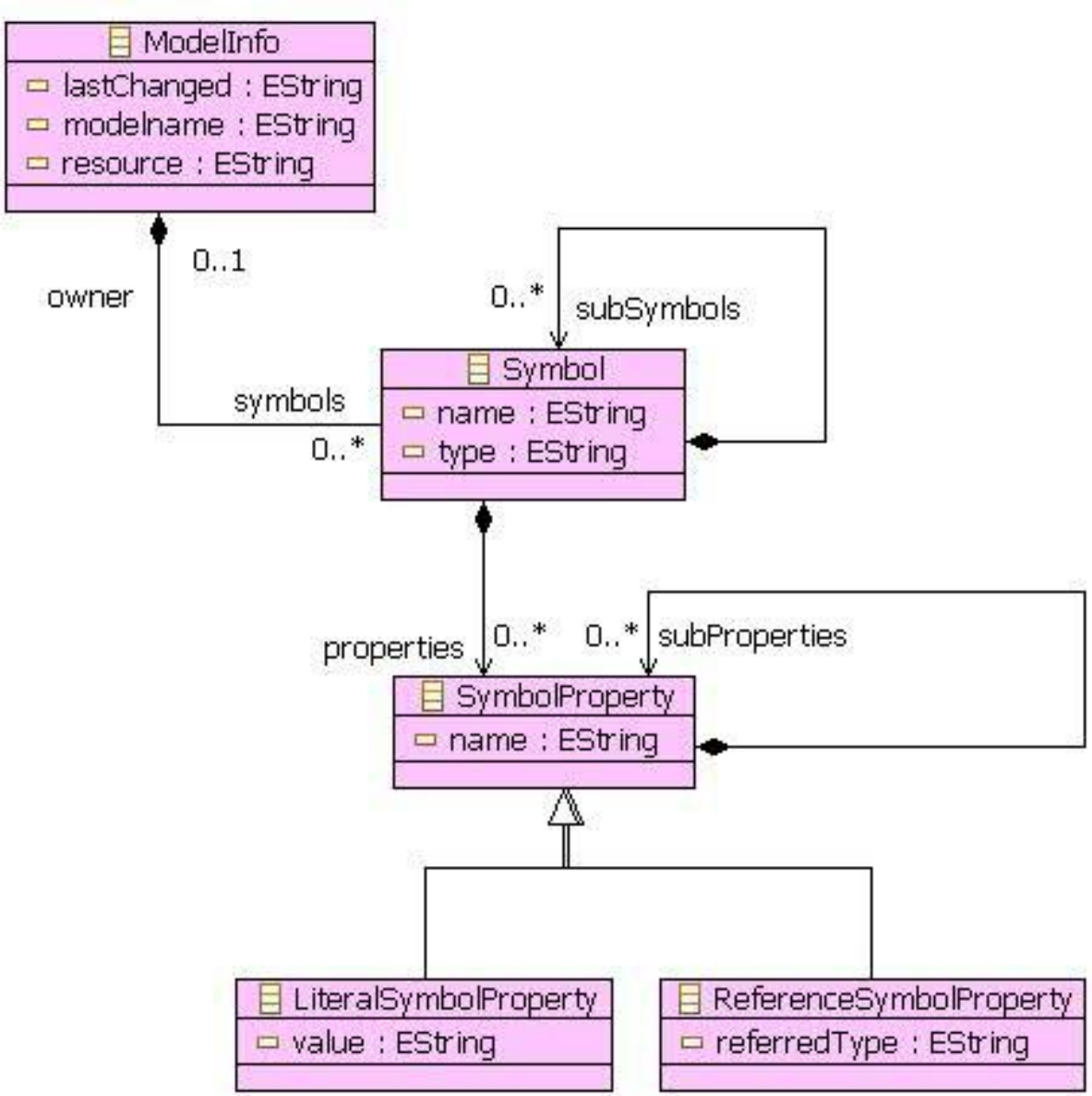
CrossX Implementation in Eclipse

Model Interface



Support for
- Cross model validation
- Code Completion in DSL
- Code generation
- Refactoring
- Propagation of model changes

Model Info structure

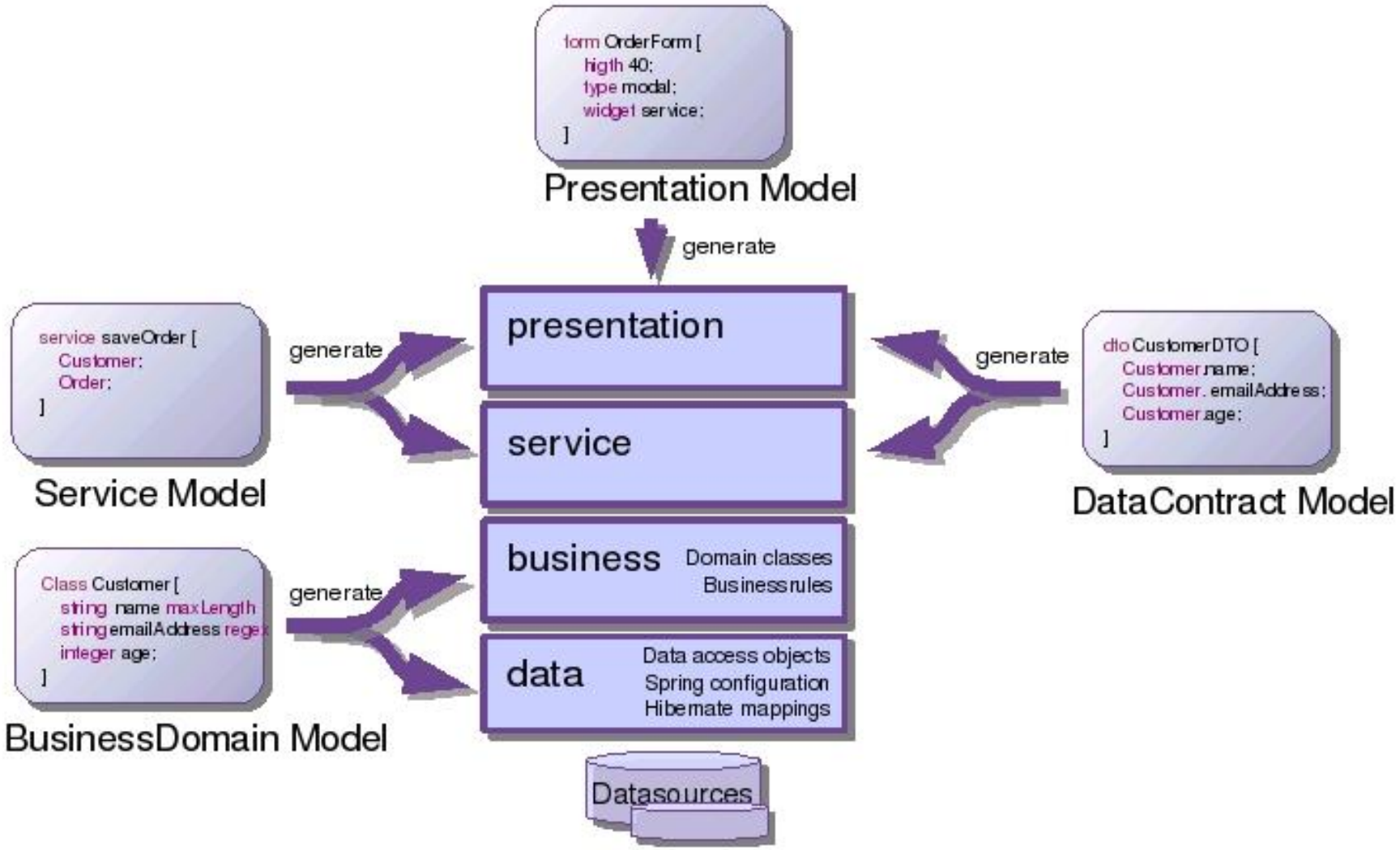


DSL Participation in CrossX

- **The DSL extends the Crossx extension point**
  - *dslName* - The readable name of the DSL
  - *dslMetamodelPackage* - The meta-model package of the DSL
  - *dslFileExtension* - The filename extension of the DSL
  - *dsl2crossxWorkflow* - The oAW workflow that will convert a DSL model unit into an crossx model unit
- **Crossx calls the workflow to get the model information at the correct time within the Eclipse build process.**

MOD4J: MODELING FOR JAVA

- **The project in which CrossX is developed**
- **Eclipse Public License**
- **www.mod4j.org and mod4j.codehaus.org**
- **Slightly delayed**
  - Original plan: now ☹
  - Credit crisis ➔ Q1 2009 ☺


form OrderForm [
high 40;
type modal;
widget service;
]
Presentation Model
generate
service saveOrder [
Customer;
Order;
]
Service Model
generate
presentation
service
generate
dto CustomerDTO [
Customer.name;
Customer. emailAddress;
Customer.age;
]
DataContract Model
Class Customer [
string name maxLength
string emailAddress regex
integer age;
]
BusinessDomain Model
generate
business
Domain classes
Businessrules
data
Data access objects
Spring configuration
Hibernate mappings
Datasources

ADVANTAGES

- **Model Unit in a file**
  - Multi user access by familiar and proven tools
  - Version management by familiar and proven tools
- **CrossX**
  - Possibility for model reuse & model libraries
  - Plug and play of new versions of libraries
  - Binary model libraries
    - just need the model interface
  - Location independence for model units
    - Never refer directly to a model unit file
  - Connecting non model artifacts
- **Scaleability**
  - Probably huge