

Data Model of the Compass/VB Analysis Tool and its Relation to KDM

Avi Yaeli, Netta Aizenbud, Jonathan Bnayahu, Nurit Dor, Sara Porat

IBM Research Labs in Haifa

Oct. 26, 2005

IBM Labs in Haifa

© 2005 IBM Corporation



Outline

- ◇ What is Compass?
- ◇ Compass Data Model
- ◇ Model 'Lessons Learned'
- ◇ Model Extension
- ◇ Summary

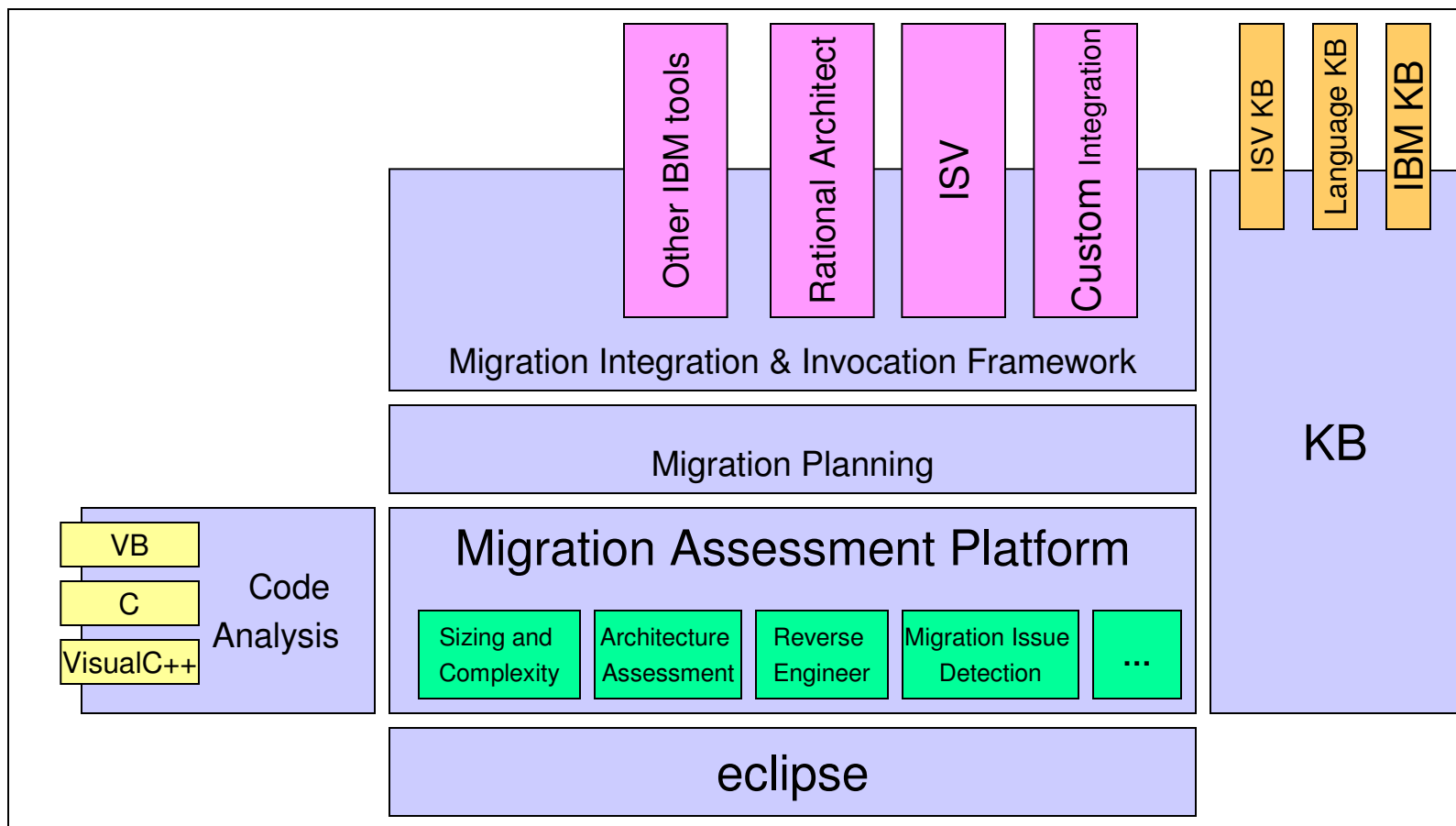


What is Compass?

- ◇ **Code Migration Planning and Assessment Workbench**
 - ◇ Understand application inventory, structure, and relationships between components
 - ◇ Understand the architecture of the application, layers, flows, and interactions between components
 - ◇ Identify obstacles and migration issues
 - ◇ Generate assessment reports
 - ◇ Help make decisions on how to migrate the application: which parts are translatable, which parts should be rewritten, etc.
- ◇ Designed to support multiple domains
 - ◇ Core, generic data model and architecture
 - ◇ Extension points for specific domains



Compass Architecture





Compass/VB Quick Fact Sheet

- ◇ Core VB analysis engine
 - ◇ Parser, AST, Symbol table, Type analysis, Call graph, etc.
 - ◇ Analysis is specific, data structure are generic
 - ◇ Form flow, Dead function
 - ◇ Used a generic engine
 - ◇ References, metrics
 - ◇ Based on a domain knowledge base
- ◇ Core Assessment Platform
 - ◇ Assessment model – generic and extensible
 - ◇ Scalable persistent store
- ◇ Knowledge Base component
 - ◇ Migration issues and resolutions, layering
 - ◇ Populated with VB and Win32 common libraries
- ◇ Eclipse based UI
- ◇ POC code migration via DPTK
 - ◇ VB Forms are automatically converted to use Eclipse Rich Client components
 - ◇ UI components and layout are converted. Stubs are created for the rest of the code



Compass Assessment Results

The screenshot displays the Compass Eclipse Platform interface for a project named 'ClockOut'. The interface is divided into several panes:

- Project Explorer:** Shows the project structure with folders for 'Application' and 'default'.
- Project Properties:** Displays 'Project Name: ClockOut' and 'Project Type: Standard EXE'.
- Files:** A table showing file counts for various extensions.
- Lines:** A table showing line counts for code, comment, and blank lines.
- Logical Elements:** A table showing counts for forms, modules, classes, and other elements.
- Migration Issues:** A table listing various migration issues with their counts and descriptions.
- Dependencies:** A table showing counts for components, references, API calls, and other dependencies.

.FRM:	16
.BAS:	2
.CLS:	1
Other:	1
Total:	20

Code:	8145
Comment:	1013
Blank:	1335
Total:	9888

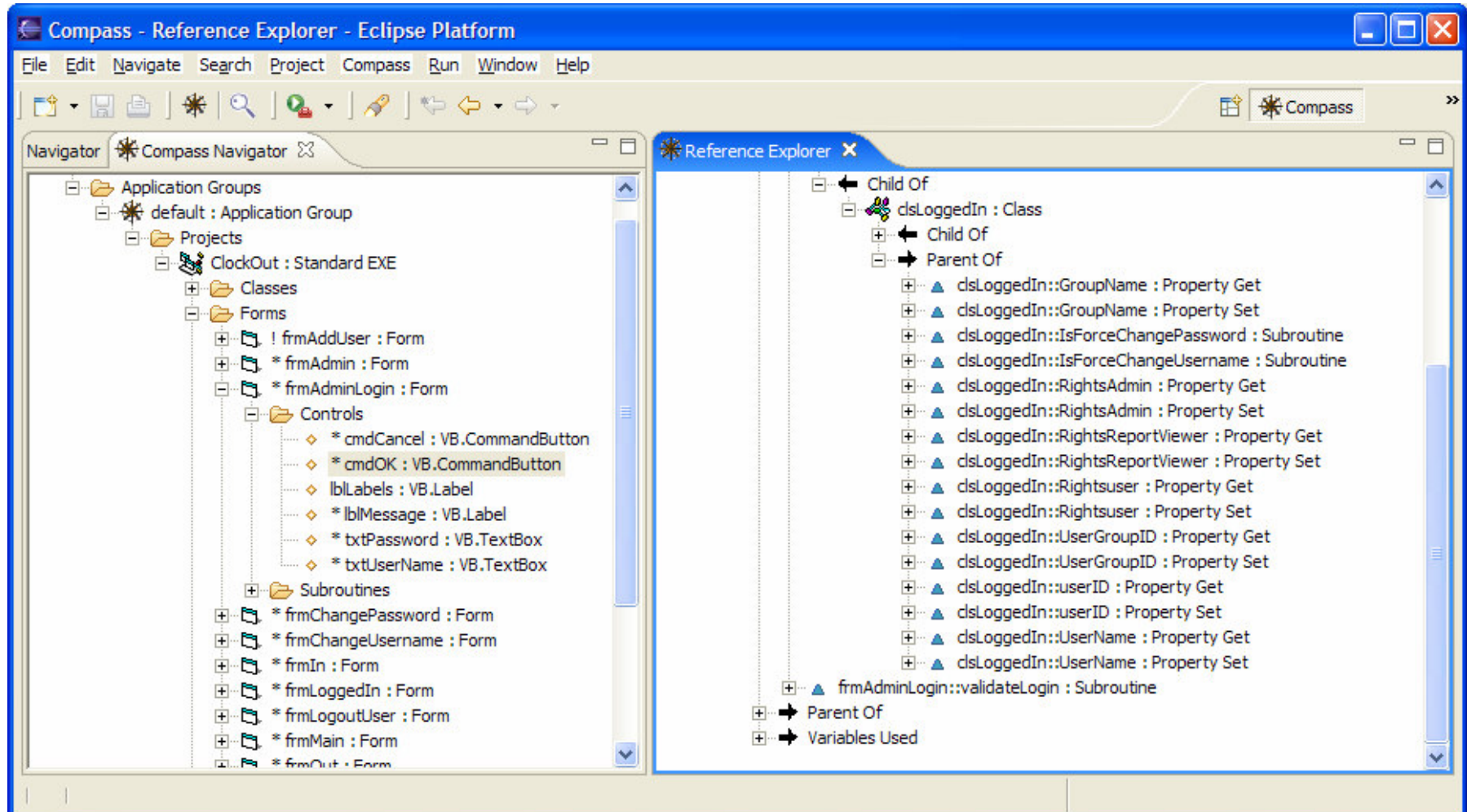
Forms:	16
Modules:	2
Classes:	1
Other:	1
Total:	20

Syntax Equiv.:	6	Syntactic and Semantic equivalent capabilities exist natively in the target platform (VB & Win32) dependent APIs/Services that have similar native equivalents.
Sem. Equiv.:	41	No syntactic equivalent but semantic equivalence can be re-produced in the target platform (VB or Win32) dependent APIs/Services with no equivalence in the source.
Minor 3rd Party:	1	Dependency on 3rd party and/or custom component/library. 3rd party component/library is not available in the target platform.
Major Platform:	0	Platform (VB or Win32) dependent APIs/Services with no equivalence in the source.
Major 3rd Party:	1	Dependency on 3rd party and/or custom component/library. No semantic equivalent exists in the target platform.
Unknown Issue:	9	Unknown migration issue
Total:	61	

Components:	16
References:	11
API Calls:	15
Other:	2606
Total:	2648

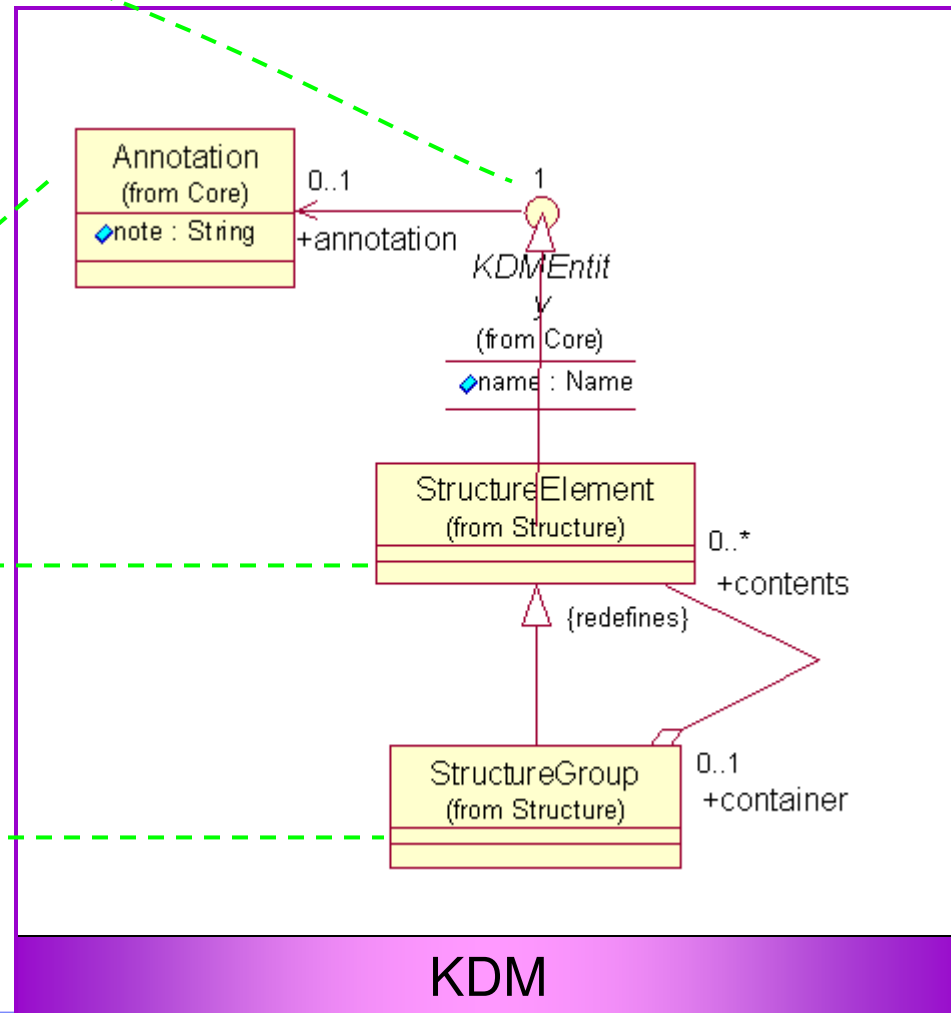
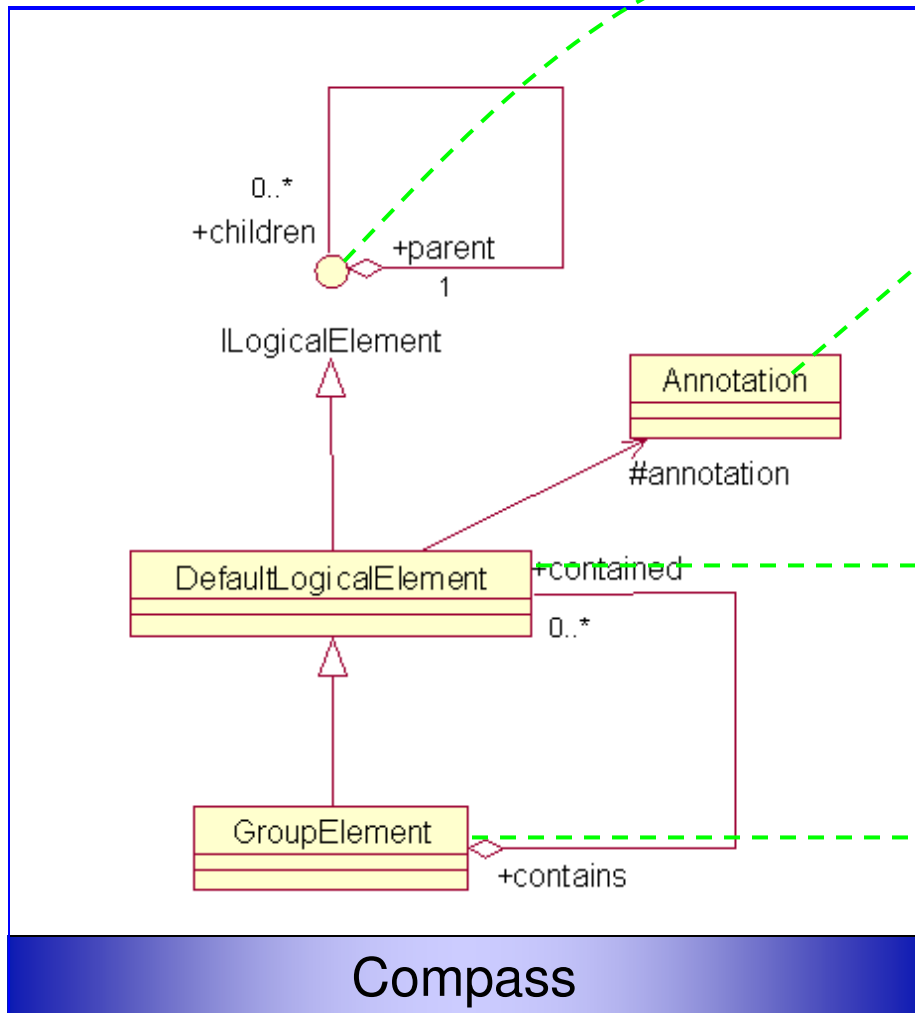


Compass Reference Explorer





Logical Model

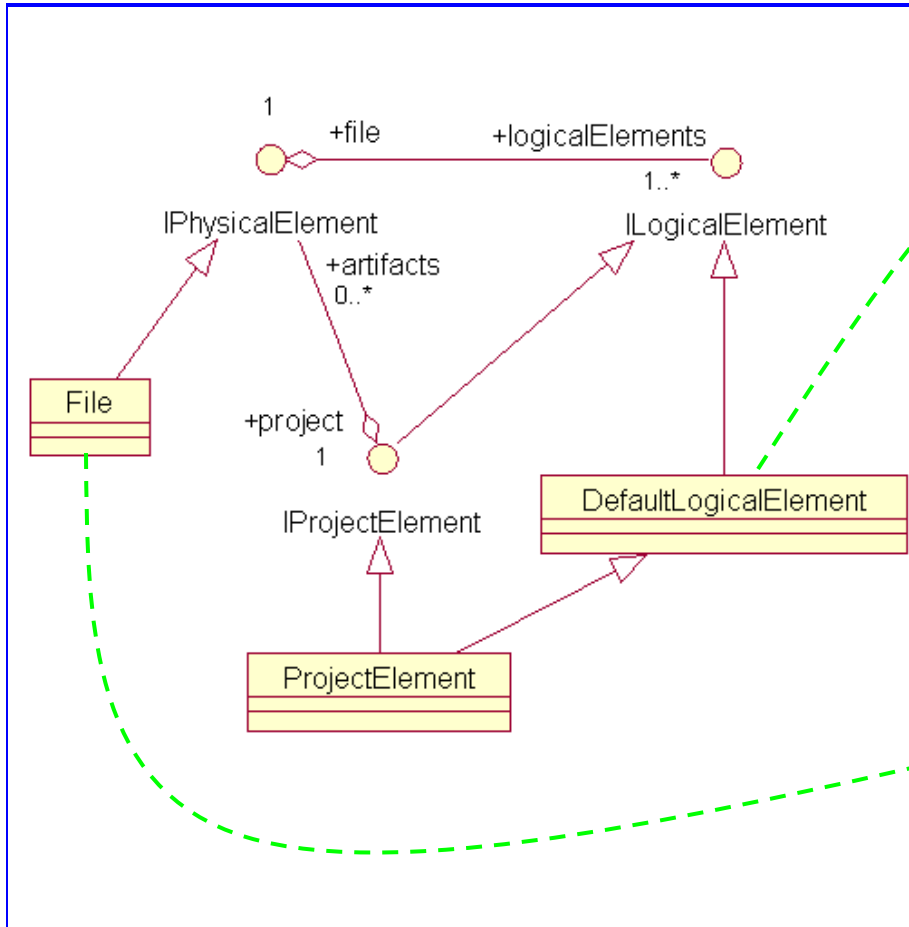


Compass

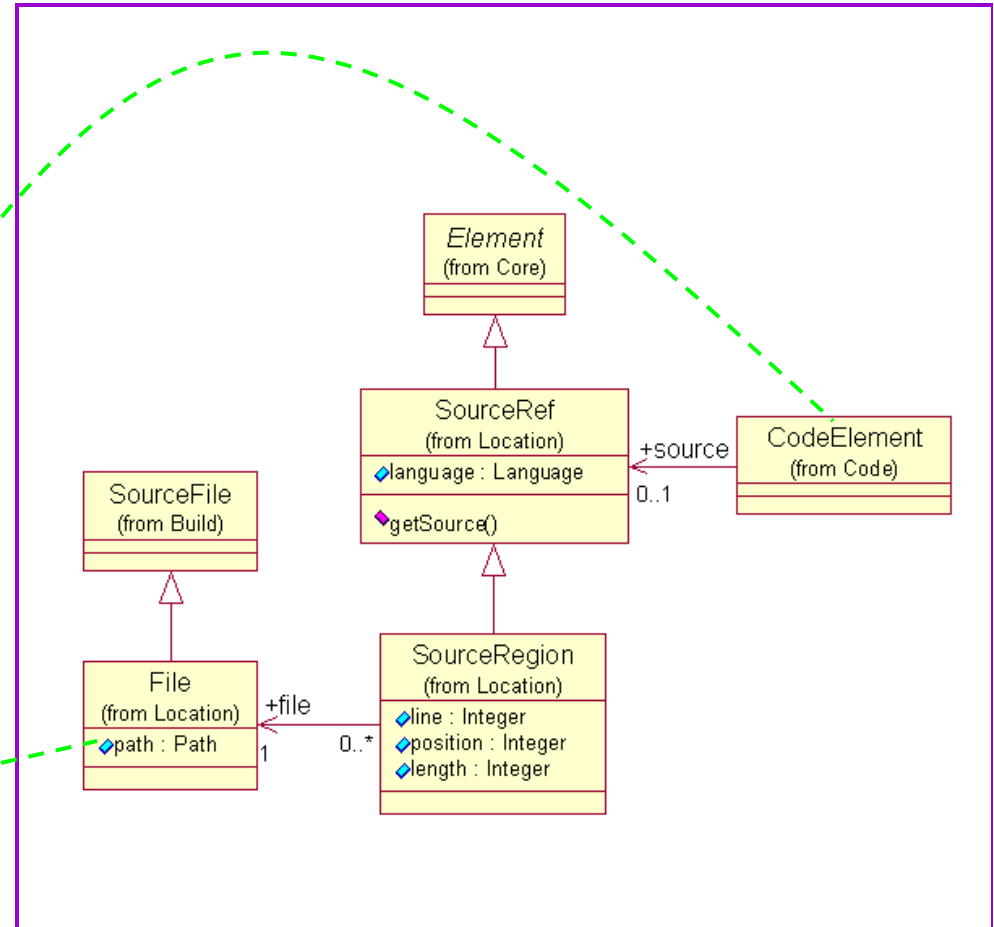
KDM



Physical Model



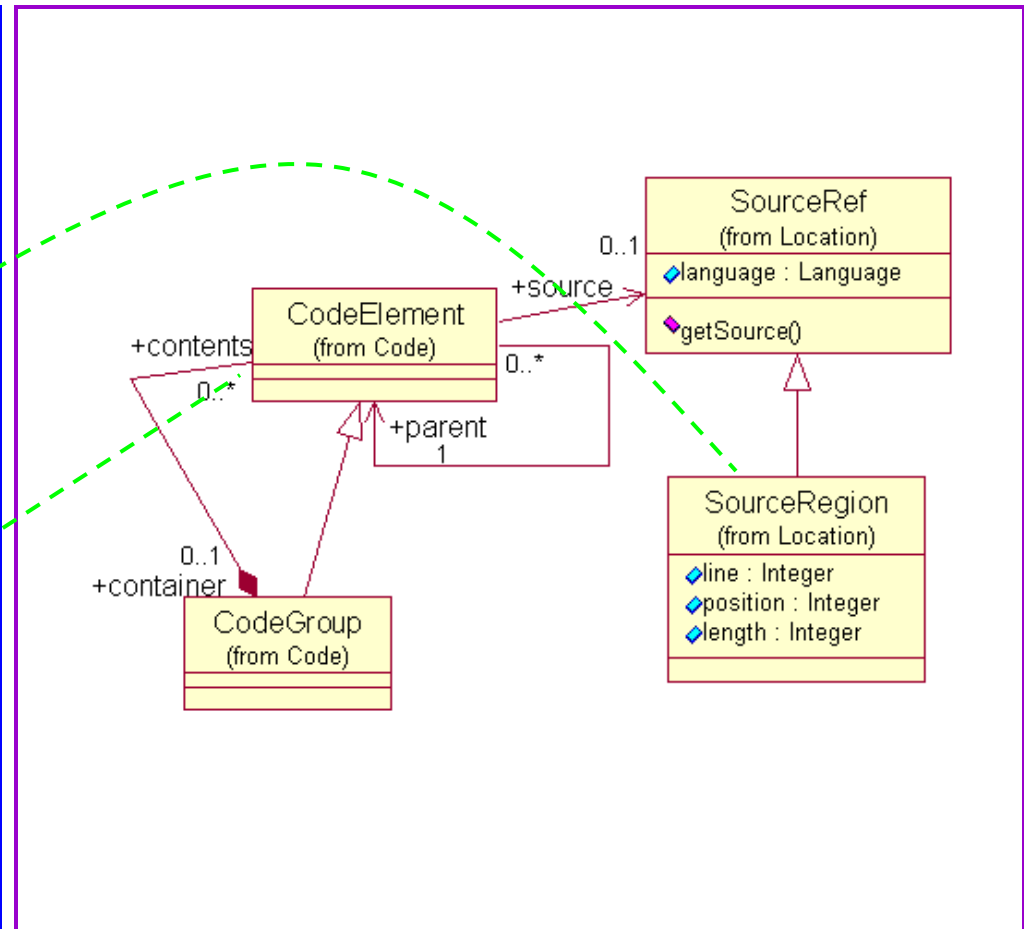
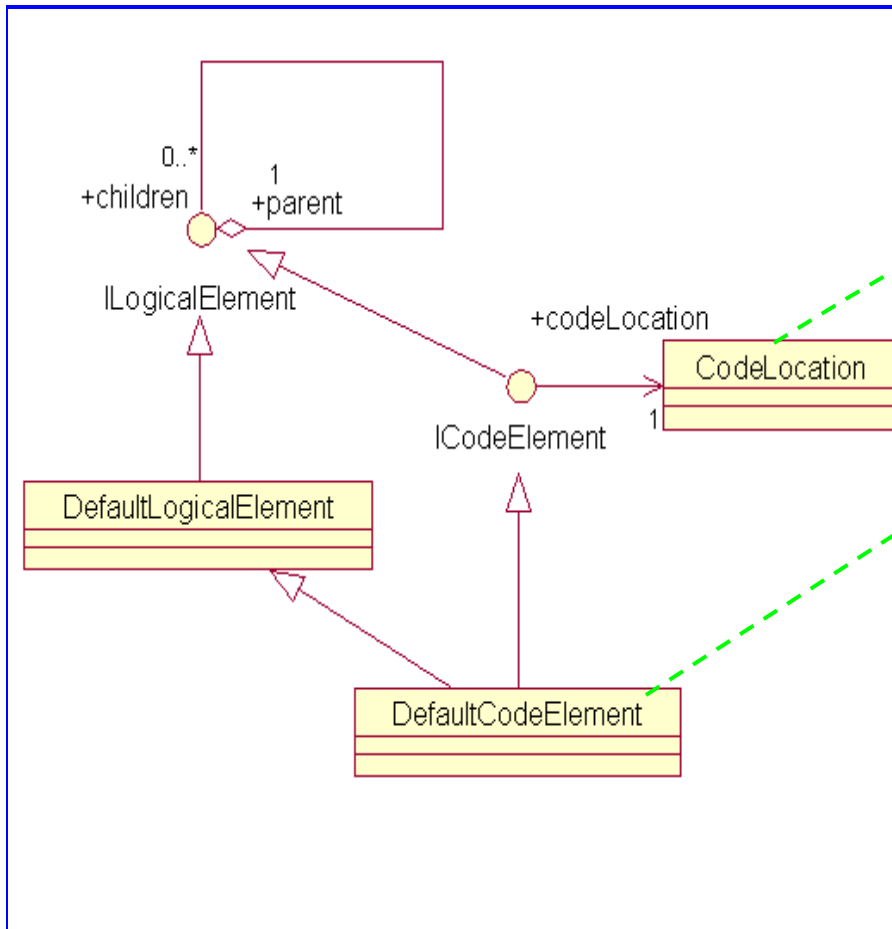
Compass



KDM



Code Model

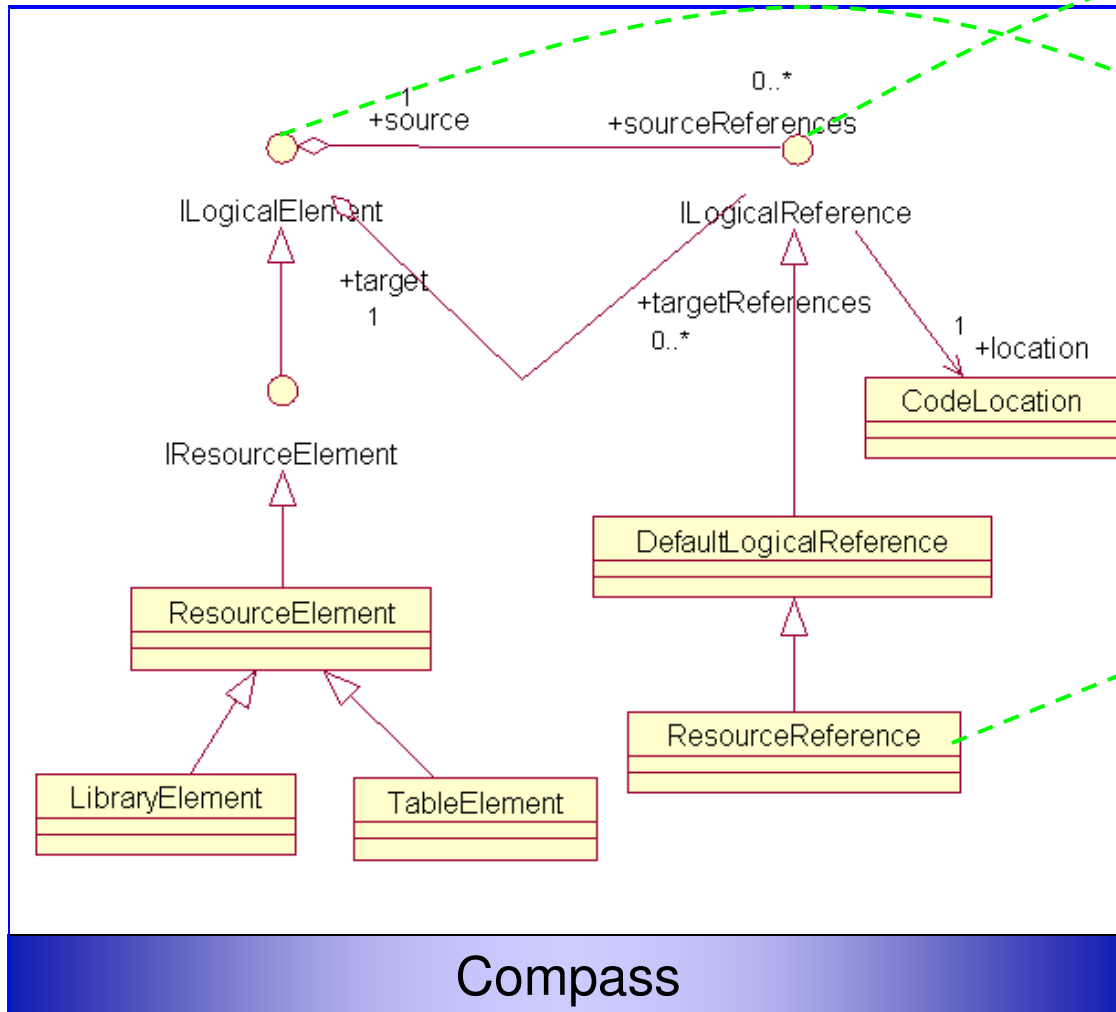


Compass

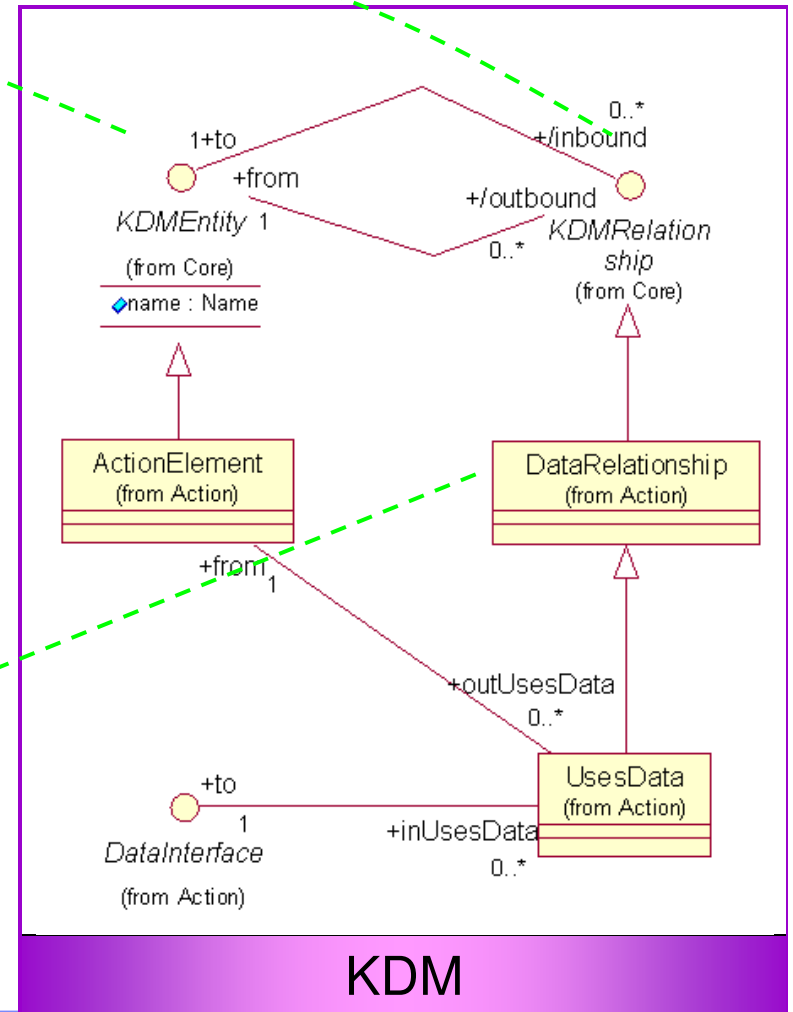
KDM



Reference Model



Compass

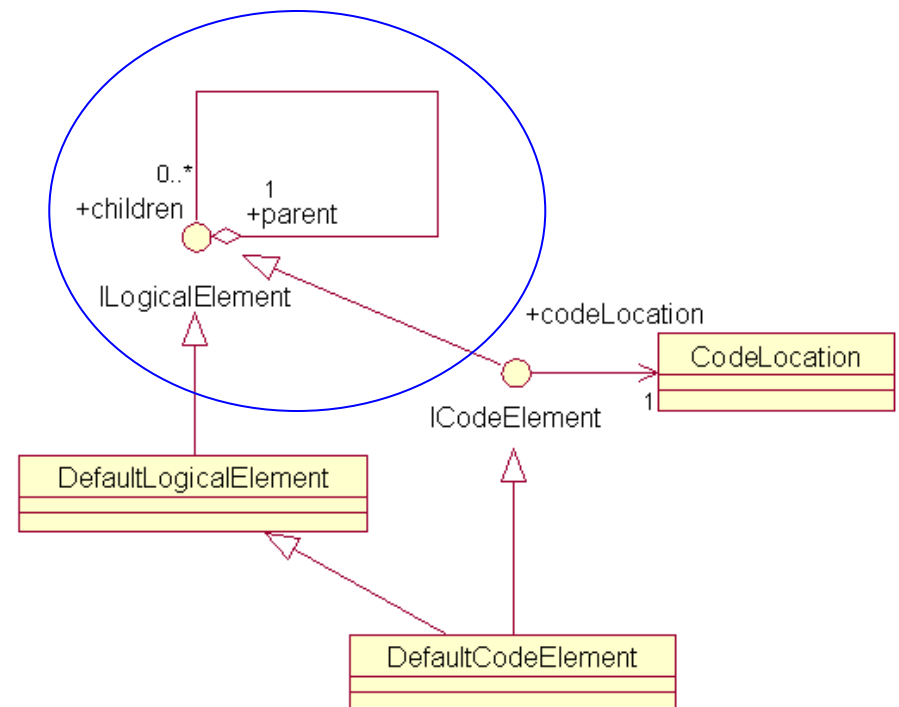


KDM



Model 'Lessons Learned'

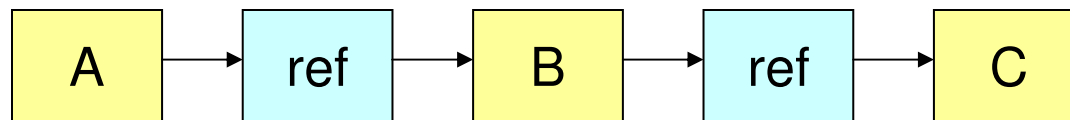
- ◆ Aggregation of information
 - ◆ 'Find all references to a resource in a project'
 - ◆ Collect data from 'child' elements all the way down the hierarchy
- ◆ How to resolve this
 - ◆ Keep links to ancestors – not realistic
 - ◆ Recursive query on a relational database



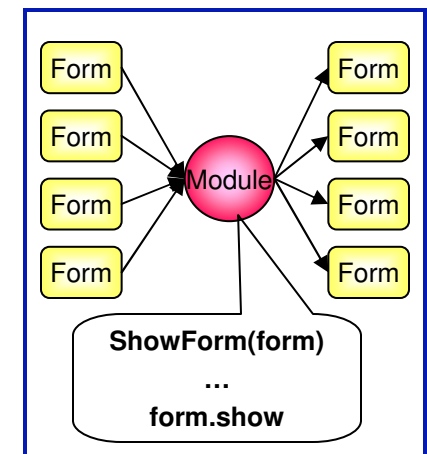


Model 'Lessons Learned' – cont.

- ◆ The semantics of transitive dependency
 - ◆ If A has a relationship to B and B has a relationship to C, does A have a relationship to C?



- ◆ Classpath dependency – yes
- ◆ Method call, UI flow – no
- ◆ Call chain – Is there a call chain from A to C?
 - ◆ not definitive, depends on the purpose of the analysis

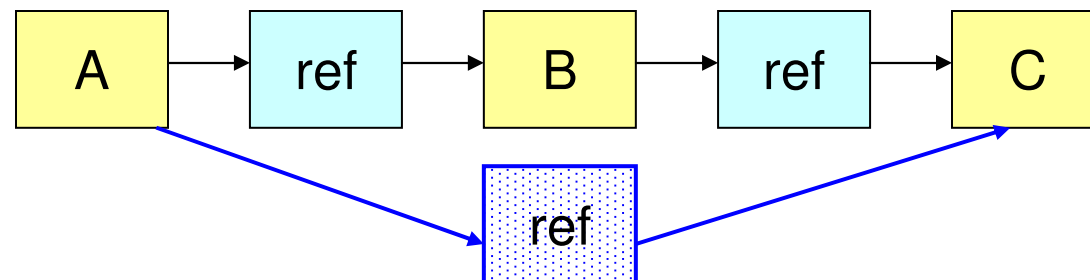


- ◆ How can this semantics be reflected in the model?



Model 'Lessons Learned' – cont.

- ◇ The semantics of transitive dependency
 - ◇ Option 1 – model the transitive reference explicitly, whenever suits the semantics
 - ◇ Not efficient – time, space

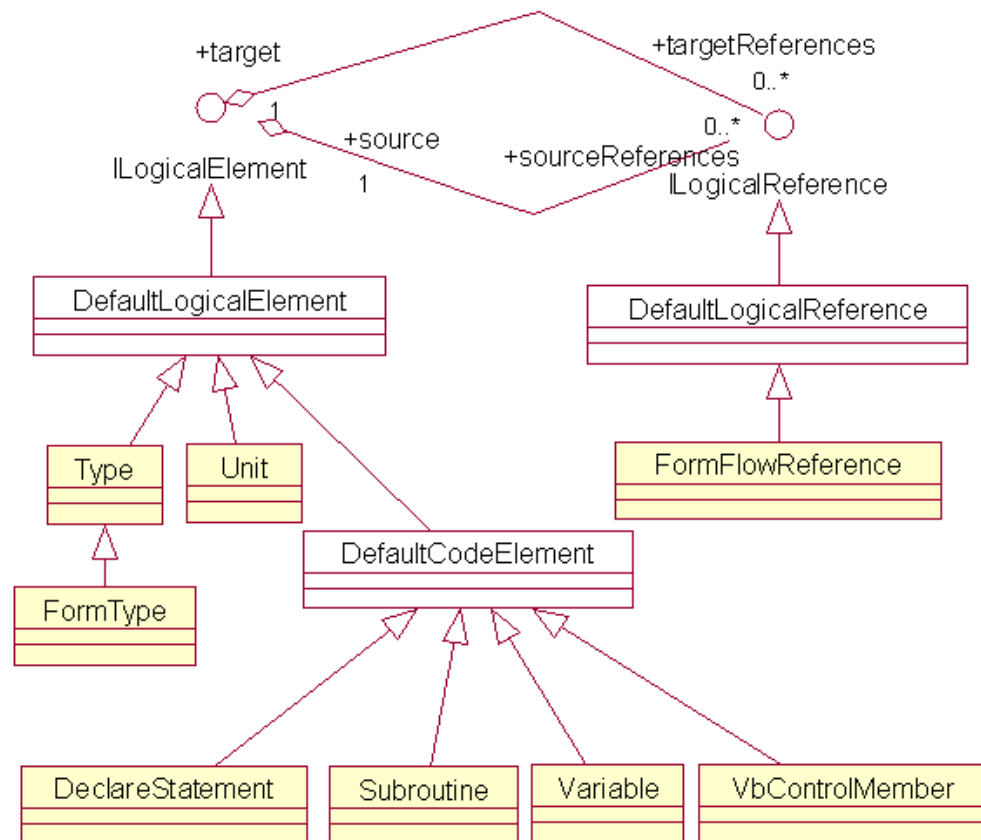


- ◇ Option 2 – add semantics to the reference – open issue
 - ◇ Add a property *Precise* for references that do not support transitive closure
 - ◇ Attach a logical operator
 - ◇ Define a partial order relation



Model Extensibility

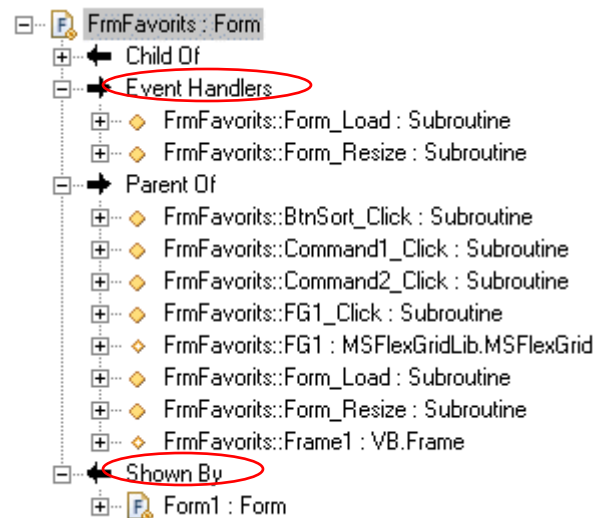
- ◆ Compass used sub-classing for extending the model
 - ◆ 'Heavy weight' approach





Model Extensibility – cont.

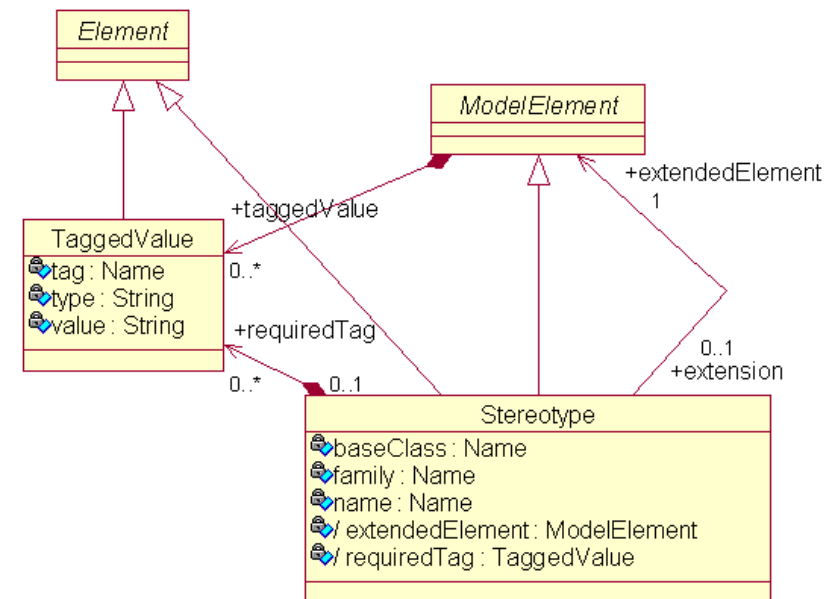
- ◆ Compass used sub-classing for extending the model
 - ◆ Fit to Eclipse extension framework
 - ◆ The domain is recognized through the extension point
 - ◆ The extending plug-in handles everything related to the extending model
 - ◆ Persistency and queries are handled generically
 - ◆ The UI used the generic model





Model Extensibility – cont.

- ◆ KDM approach is 'light weight'
- ◆ An extension for a specific language includes creation of stereotypes for all entities and relations in the given language
 - ◆ Creating a complete family of stereotypes



```
<stereotype name="JavaClass" baseClass="ClassUnit" family="Java">
  <TaggedValue tag="isAbstract" type="Boolean"/>
  <TaggedValue tag="isPublic" type="Boolean"/>
  <TaggedValue tag="isFinal" type="Boolean"/>
</stereotype>
```



'Light weight' vs. 'Heavy weight' approaches

Light Weight	Heavy Weight
Pros	
<ul style="list-style-type: none">◆ Do not require reposting the standard, as they do not change the XMI◆ Possible to fully differentiate between entities and relations for different languages◆ More uniform model and queries◆ Every specialization is both semantically and syntactically compatible with its parent concept	<ul style="list-style-type: none">◆ Allows definitions of new concepts in the most direct way possible<ul style="list-style-type: none">◆ If a concept is missing from KDM, just add it as an additional class◆ Less extension work◆ More straightforward queries◆ More convenient for tool vendor specific work
Cons	
<ul style="list-style-type: none">◆ Not always possible to map a domain specific concept to a KDM concept in a precise and meaningful way◆ Creating a complete set of extensions is a lot of work◆ The KDM Queries for browsing through the model are slightly more complex◆ The output models are bigger	<ul style="list-style-type: none">◆ Require reposting the standard if the extension should be used for interoperability◆ More difficult to handle multi-language models◆ Less uniform queries◆ Rather expensive to produce and maintain complete tool chains for custom language models



Summary & Conclusions

- ◆ Developing common models have been a need for almost any Application Mining vendor who wants to support multiple languages or interoperate with 3rd party tools.
- ◆ Compass has developed its own common model
 - ◆ **Very close to the KDM standard**
- ◆ This exercise builds more confidence about the KDM standard
 - ◆ **Tool vendors would be able to map their internal representations to KDM**
- ◆ There are still issues that needs to be addressed with the usability, scalability of the model
- ◆ Extension approach – need to get feedback from ‘field’ experience