Knowledge Discovery Meta-model: Tutorial

ADM workshop, 24th October 2005
Washington DC

Presenter:
Dr. Nikolai Mansurov
Chief Scientist, Klocwork

2nd joint revised submission published 3-04-2005

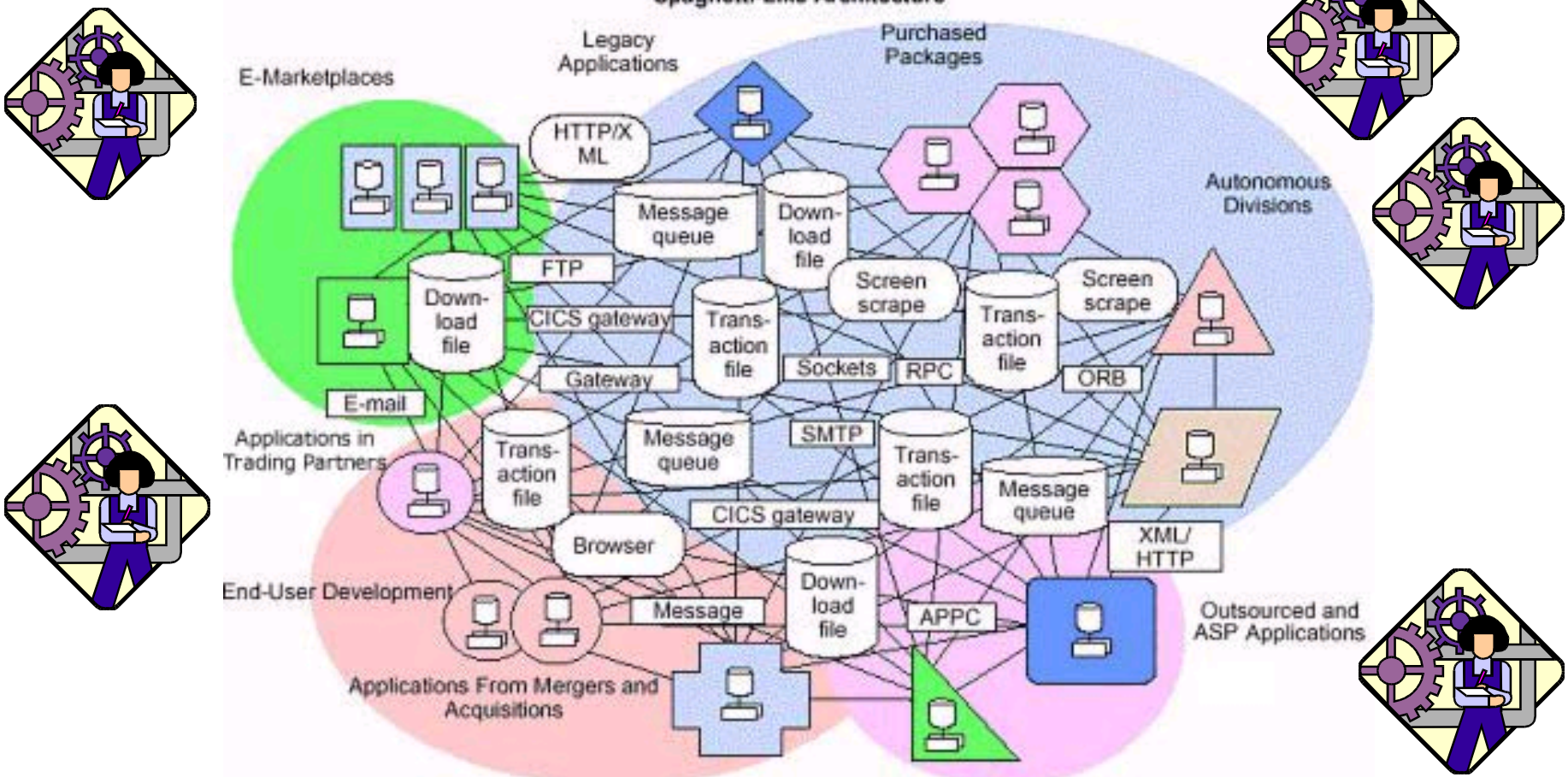
Supporters:

Outline of the tutorial

1. Introduction
2. Core package
3. System and environment packages
4. Code and Action packages
5. Data package
6. Logical, Run-time, Build packages
7. Conceptual and Behavior packages
8. UI package

Introduction

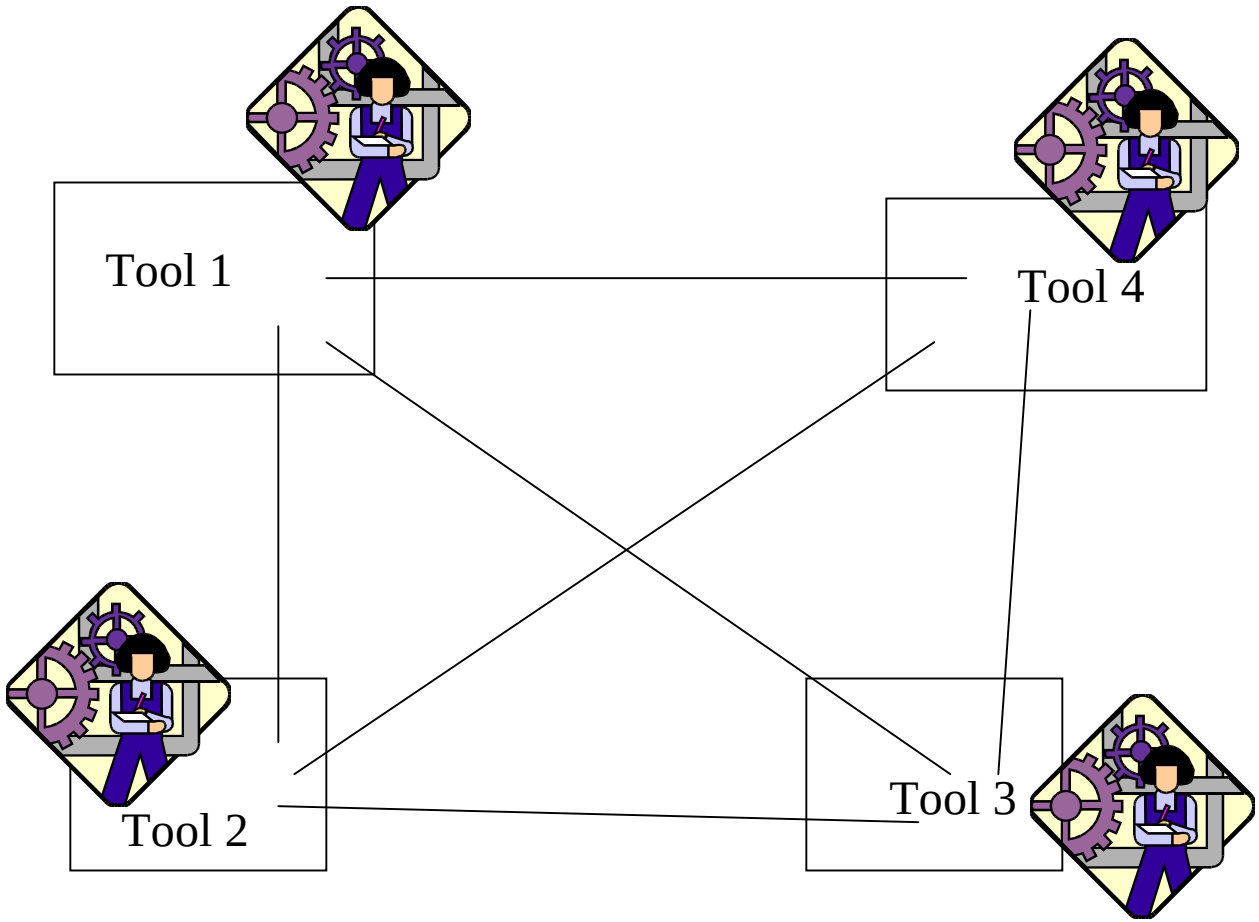
Business case for KDM



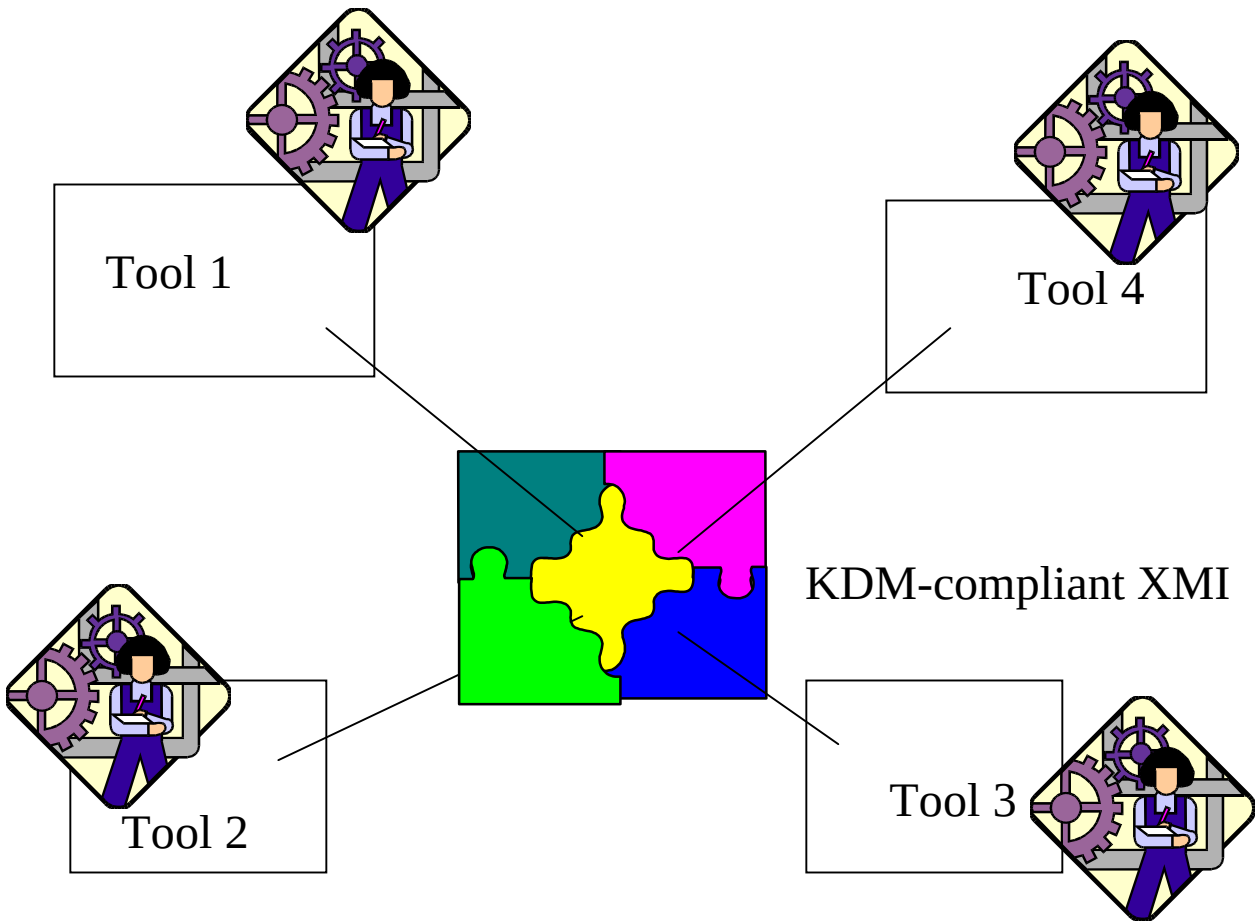
Business case for KDM

- Existing software: complex, multiple technologies; multiple vendors;
- Need to extract knowledge in order to:
  - understand,
  - maintain,
  - improve,
  - transform,
  - modernize
- Need cooperation between vendors – starting with common knowledge representation

KDM as tool interchange format



KDM as tool interchange format



What is KDM ? (excerpt from RFP)

- The RFP solicits proposals for a Knowledge-Discovery Meta-model (KDM) for exchanging information related to transformation of existing software assets. Specifically, the proposal seeks a common repository structure for representing information about existing software assets and its operating environments.
- KDM represents the structure of the existing software and its related artifacts. It provides the ability to document existing systems, discover reusable components in existing software, support transformations to other languages and MDA, or enable other potential transformations.
- The meta-models will enable exchange of information about existing software artifacts among different tools. This will enable vendors that specialize on certain languages, platforms or types of transformations to deliver customer solutions in conjunction with other vendors.

What is KDM ? (summary)

- Meta-model for representing information about existing software assets
- Includes software and its operating environment
- Exchange of information between tool vendors

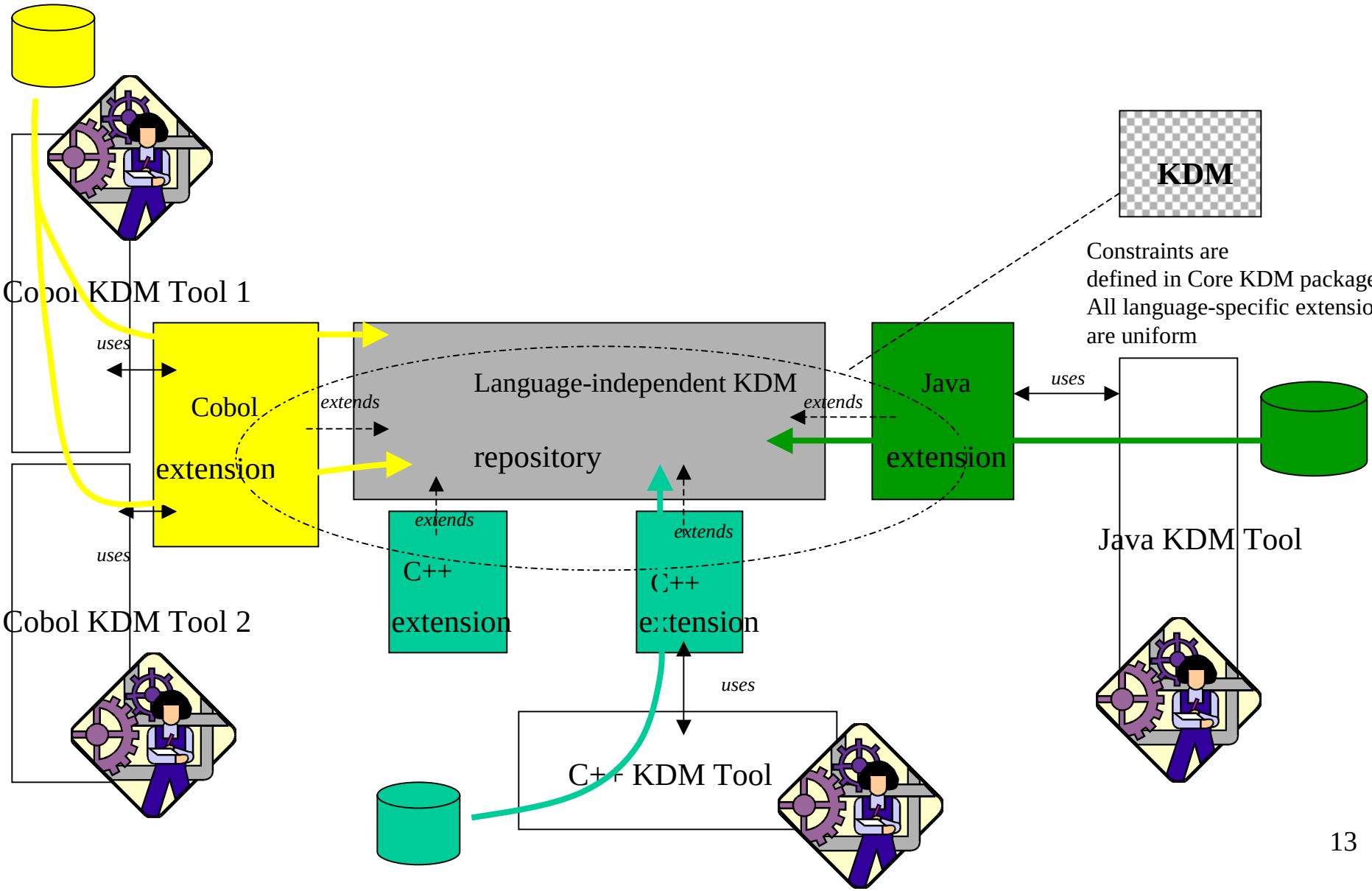
What exactly is KDM ? (excerpt from RFP)

- KDM represents the principal artifacts of existing software as entities (the structural elements, the “things”), their relationships and attributes
- KDM spans multiple levels of abstraction:
  - Implementation
  - Design
  - Architecture
  - Business rules
- KDM provides traceability to artifacts
- KDM is independent of implementation language and platform
- KDM can represent multiple heterogeneous systems
- KDM is extensible

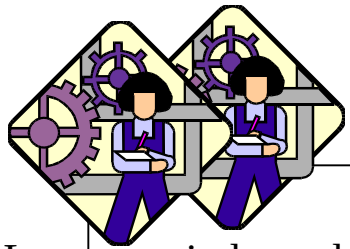
What exactly is KDM ? (summary)

- Represents existing software as Entities, Relationships and Attributes
- Language-independent yet extensible
- Spans multiple abstractions yet traceable back to artifacts
- Focuses on structure of existing software to provide context for modernizations

KDM in action: extraction



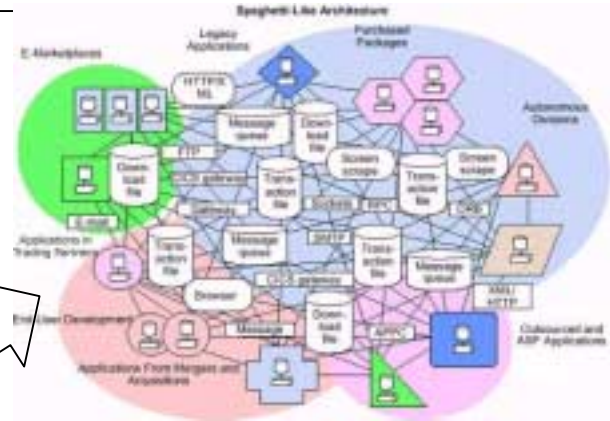
KDM in action: integration



Language-independent KDM Tool

modifies

integrates



Cobol extension

extends

Language-independent KDM repository

extends

Java extension

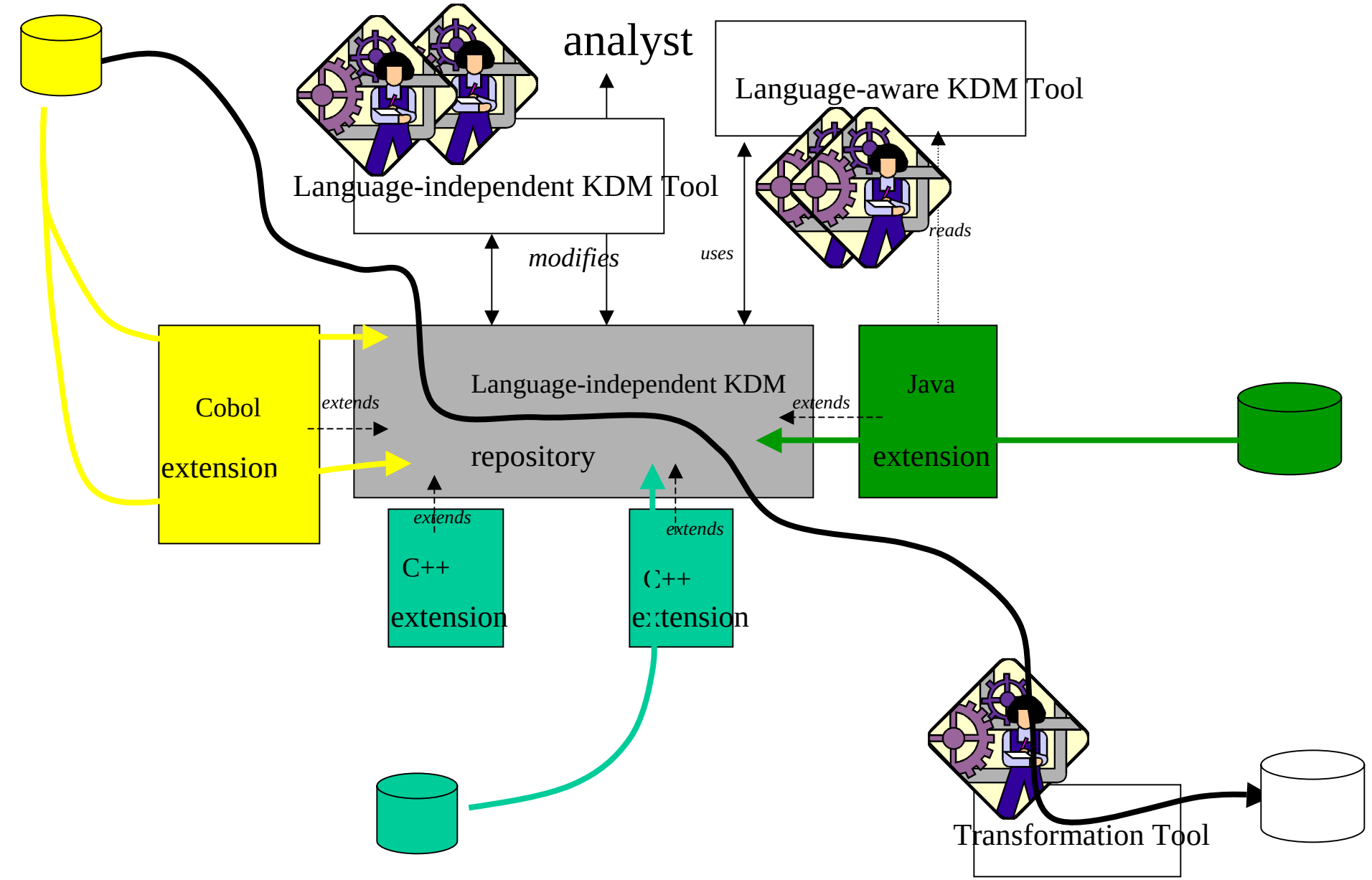
extends

C++ extension

extends

C++ extension

KDM in action: analysis

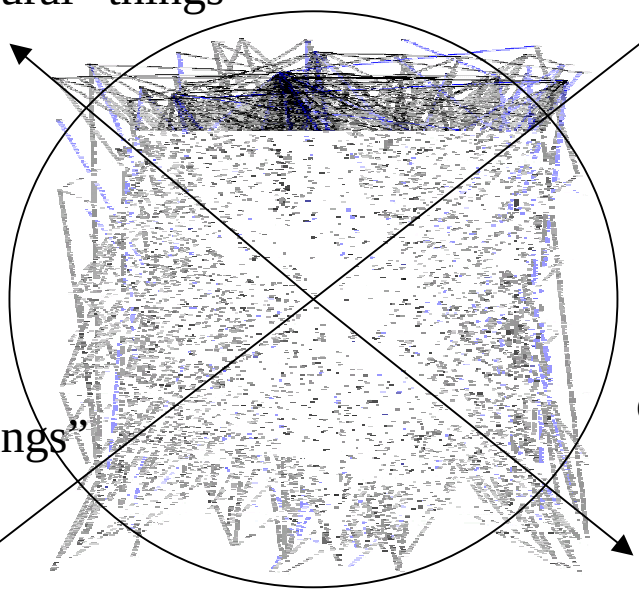


Where KDM adds value?

- Information exchange between vendors
- Architectural context for
  - Understanding software
  - Modularization, untangling “hairballs”
  - Discovering reusable components
  - Inventory, estimations of effort
  - Modernization
- Architecture management
  - Before modernization
  - After modernization
- Application Portfolio Management
- Transition into MDA

Knowledge “dimensions” in KDM and separation of concerns

Conceptual architecture



Domain model

Approximates modeling world from the ground up

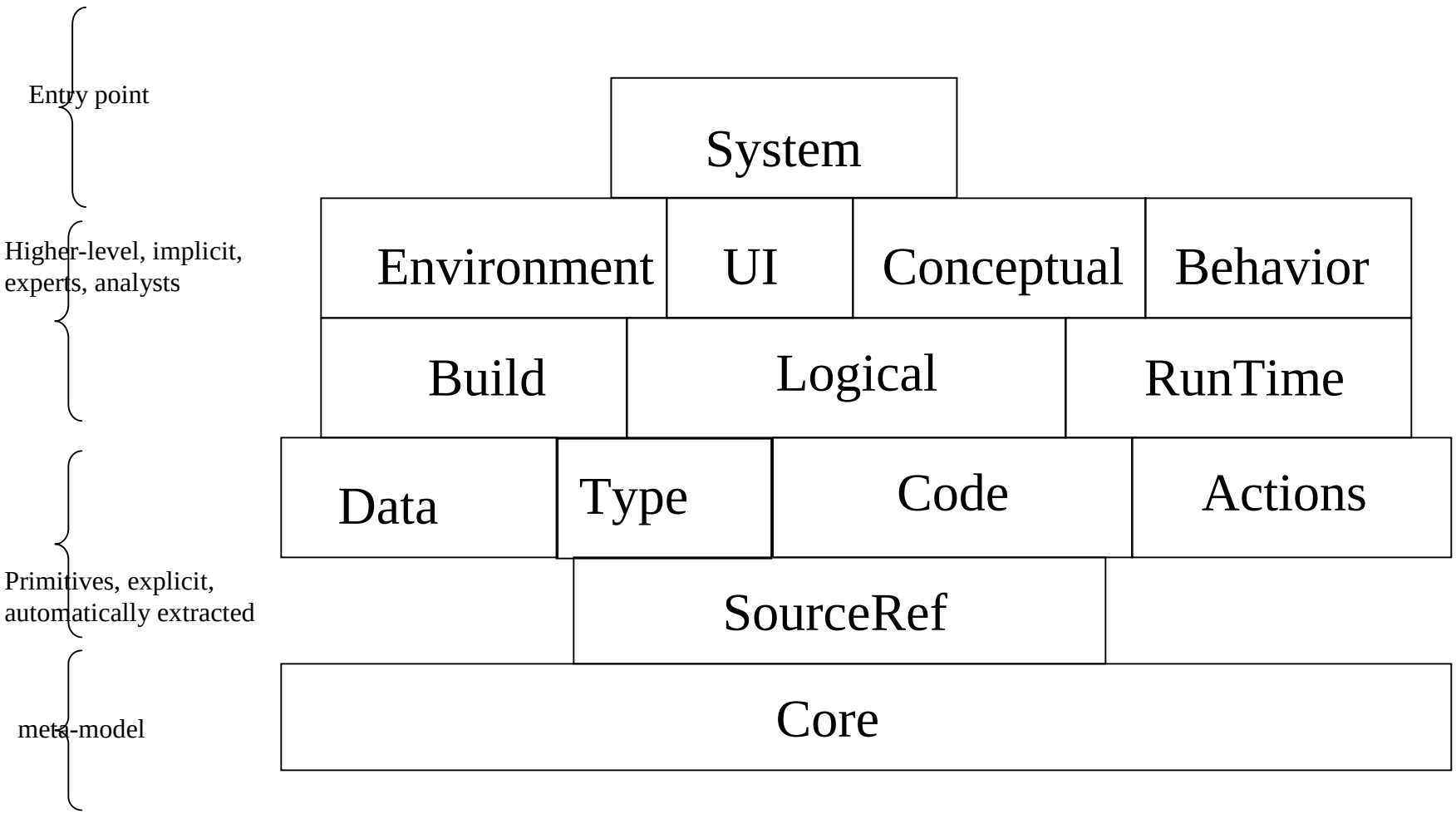
Requirements,
Features,
Use Cases

Based on Architecture Views

How KDM is designed ?

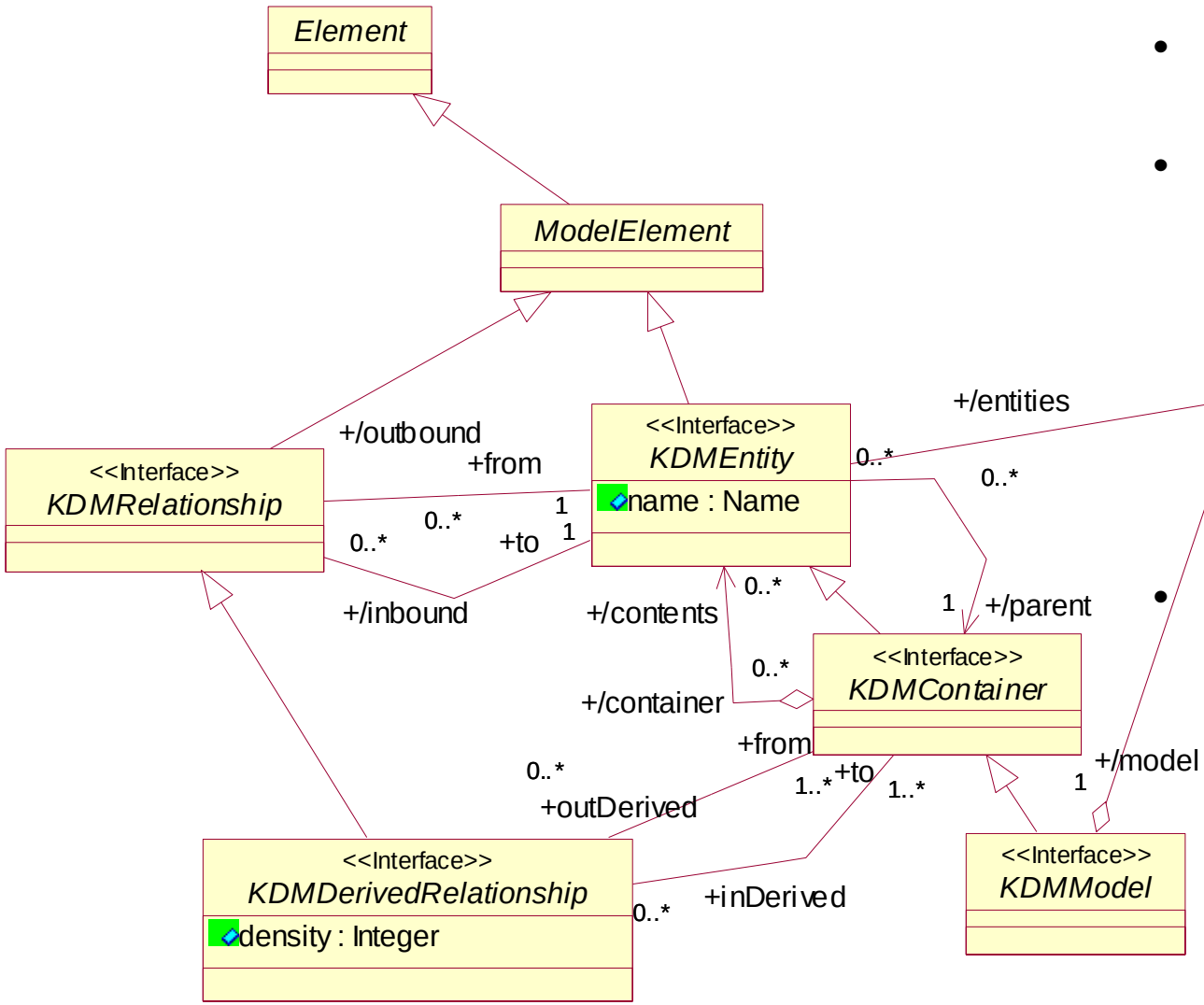
- KDM is partitioned into several packages
- Each package represents software artifacts as entities and relations
  - Code, Data and Action entities
  - Logical, RunTime, Build entities
  - UI entities
  - Conceptual entities
  - Behavior entities
  - Environment entities
- Since KDM focuses on representing structures, most KDM entities are *containers* for other entities
- KDM can represent multiple container hierarchies
- Uniform mechanism of *derived relations* bottom-up through container hierarchies to address scalability of abstractions

Structure of KDM packages



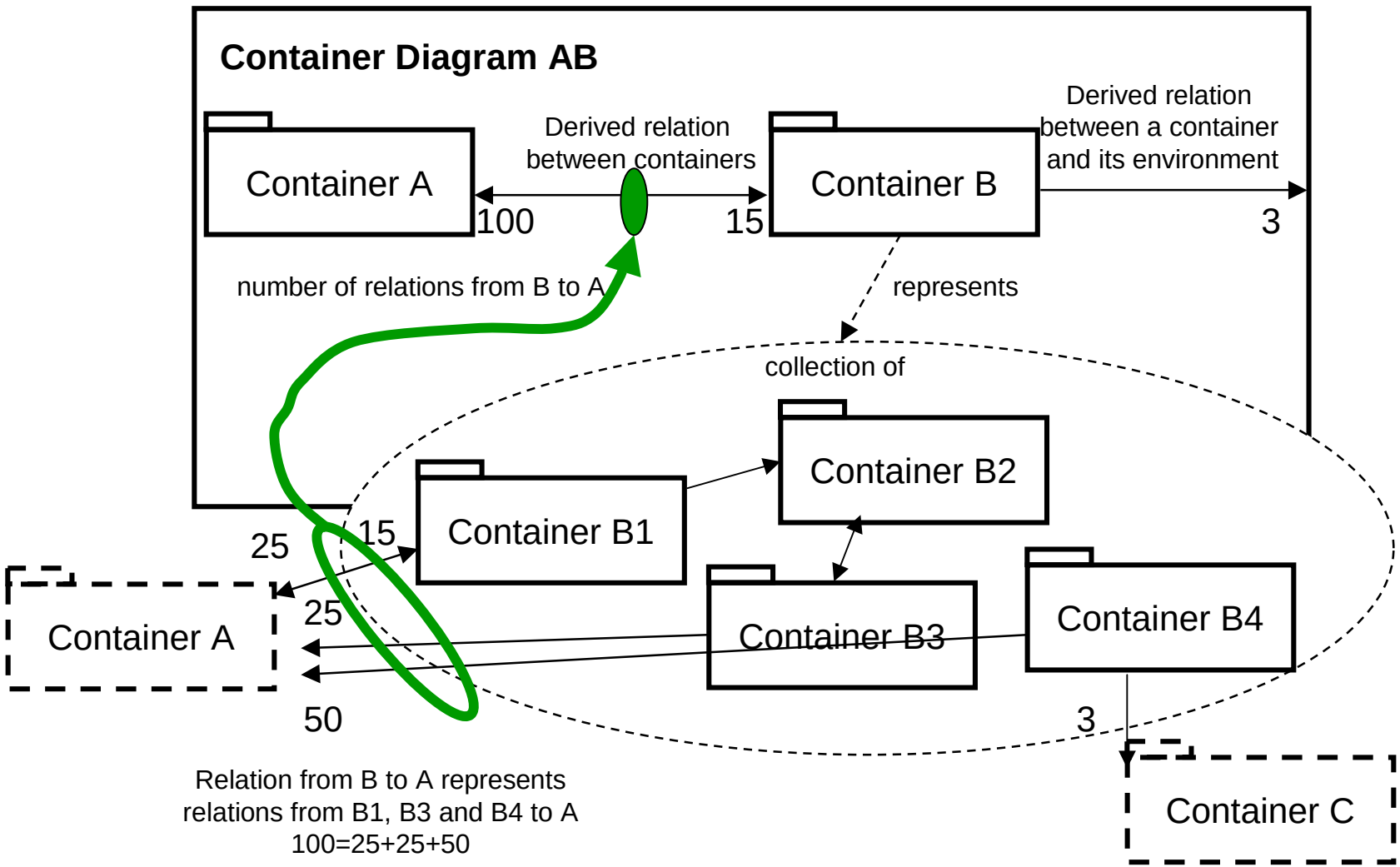
Core Package

Core KDM package



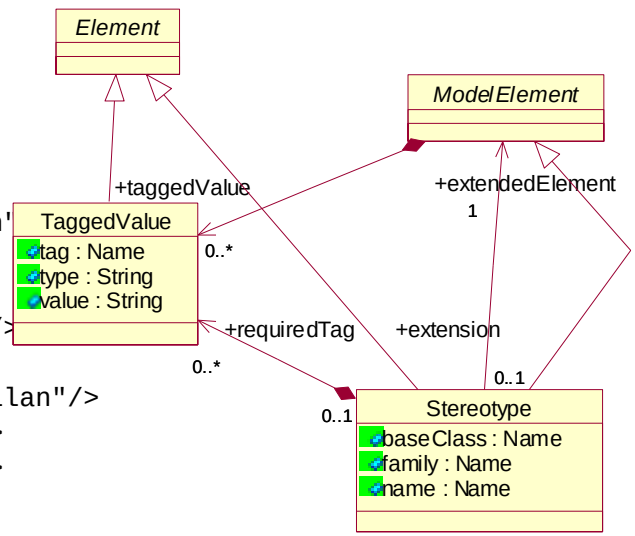
- Core package is used by all other KDM packages
- Core package defines key KDM concepts and constraints
  - KDMEntity
  - KDMRelationship
  - KDMContainer
  - KDMModel
- Core package defines KDM light-weight extension mechanism
  - Stereotype
  - TagValue

Derived relationships



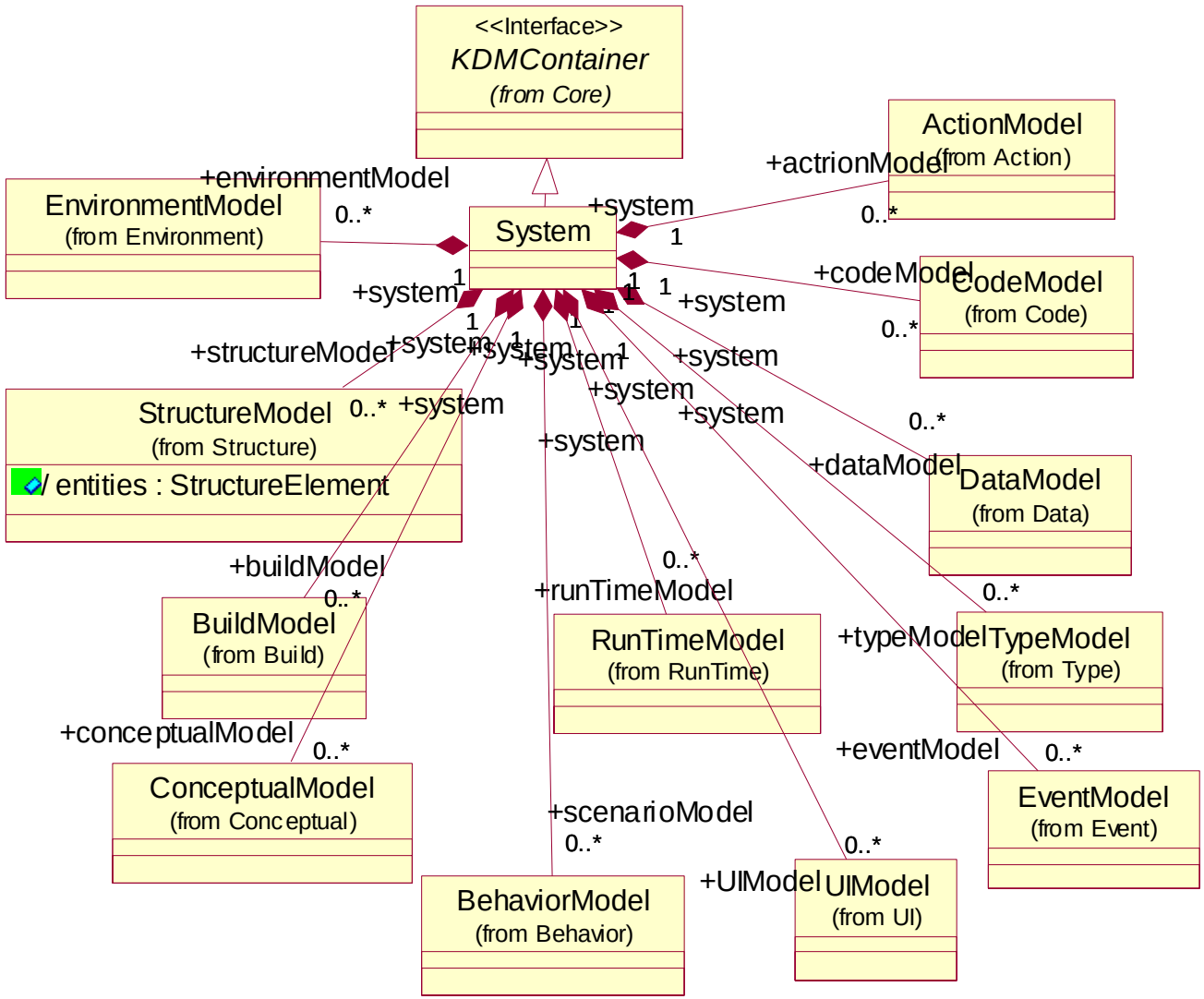
Light-weight extension example

```
<?xml version="1.0" encoding="UTF-8"?>
<null:System xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:null="null">
  <Executable name="AMilanProgram" extension="//@stereotype.0" >
     <TaggedValue tag="reenterable" value="false"/>
</Executable>
  <CallableUnit name="aFunc" parent="//@Executable.0"/>
  <Data name="aLocalVar1" extension="//@MetaEntity.11" parent="//@CallableUnit.0"
           extension="//@stereotype.4"/>
  <Data name="aParam1" kind="//@MetaEntity.12" parent="//@CallableUnit.0"
            extension="//@stereotype.5"/>
  <Data name="aParam2" kind="//@MetaEntity.12" parent="//@CallableUnit.0"
             extension="//@stereotype.5"/>
  <DataType name="int" kind="//@MetaEntity.8" parent="//@Executable.0"
              extension="//@stereotype.2"/>
  <Data name="aVar" kind="//@MetaEntity.7" parent="//@Executable.0"
              extension="//@stereotype.1"/>

  <stereotype baseClass="Executable" name="MilanProgram" family="Milan'
        <TaggedValue tag="reenterable" type="Boolean"/>
       </stereotype>
  <stereotype name="MilanVariable" baseClass="Global" family="Milan" />
  <stereotype name="MilanType" baseClass="DataType" family="Milan"/>
  <stereotype name="MilanFunction" baseClass="CallableUnit" family="Milan"/>
  <stereotype name="MilanLocal" baseClass="LocalData" family="Milan"/>
  <stereotype name="MilanParam" baseClass="ParamData" family="Milan"/>
  <stereotype name="UsesType" baseClass="HasType" family="Milan"/>

</null:System>
```



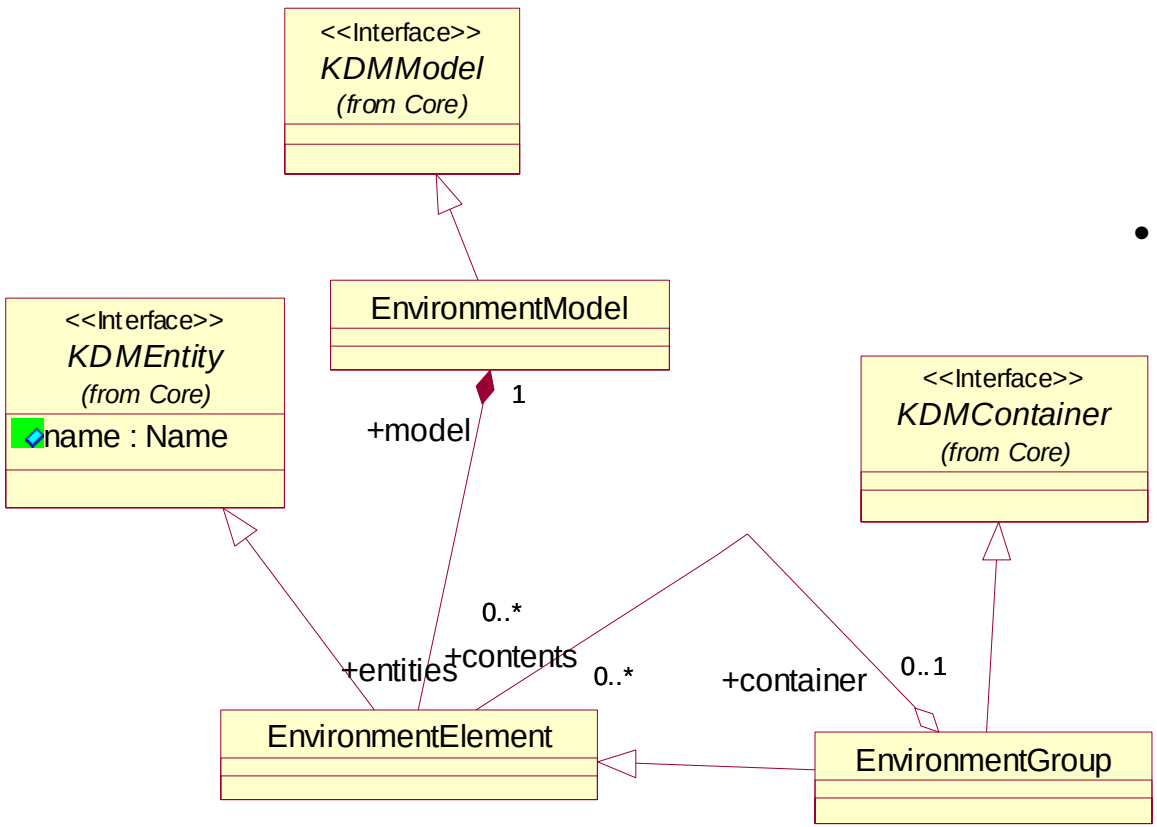
System and Environment packages

System



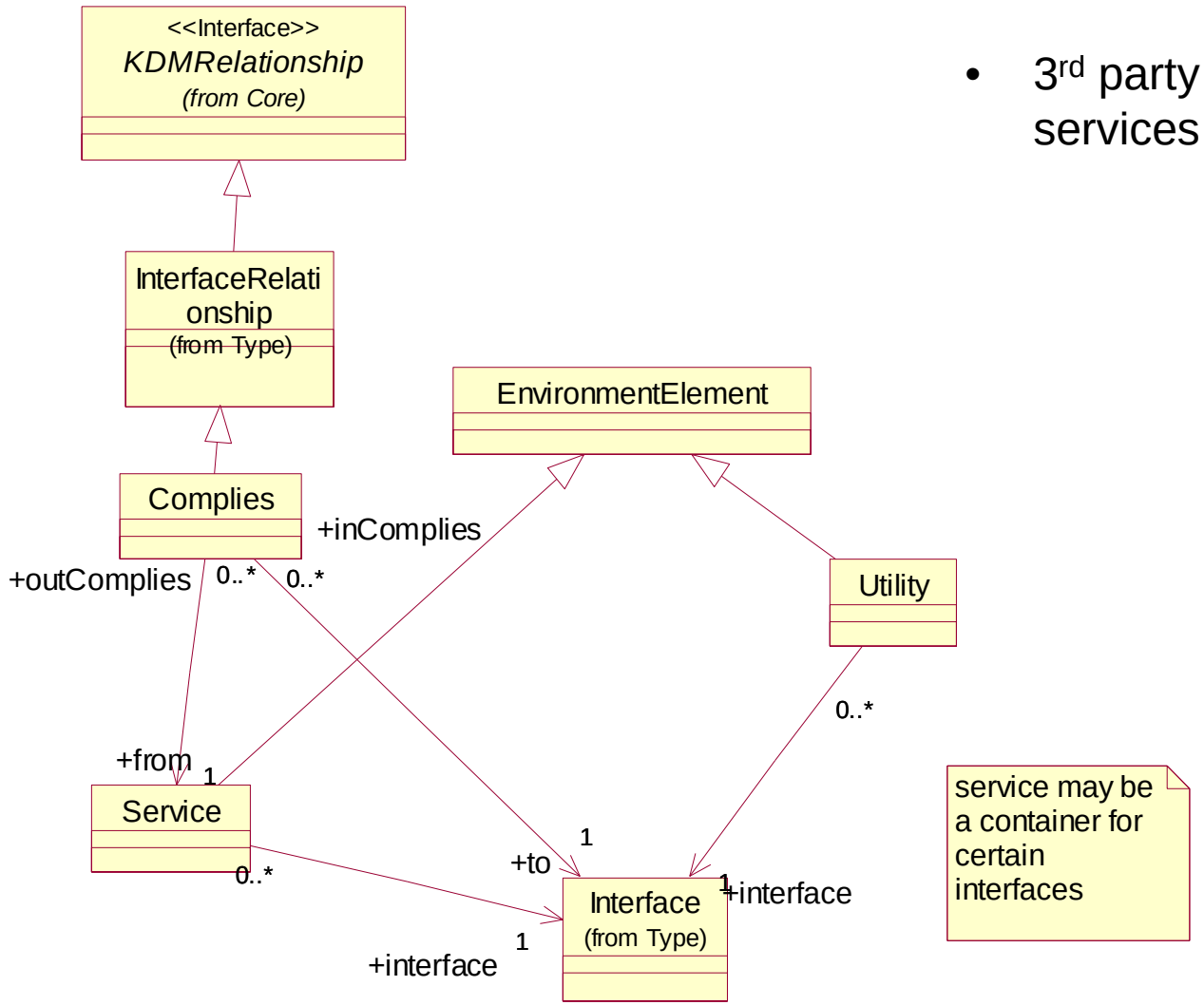
- System is a collection of various models
- System is an entry point to various facets of knowledge

Environment Package



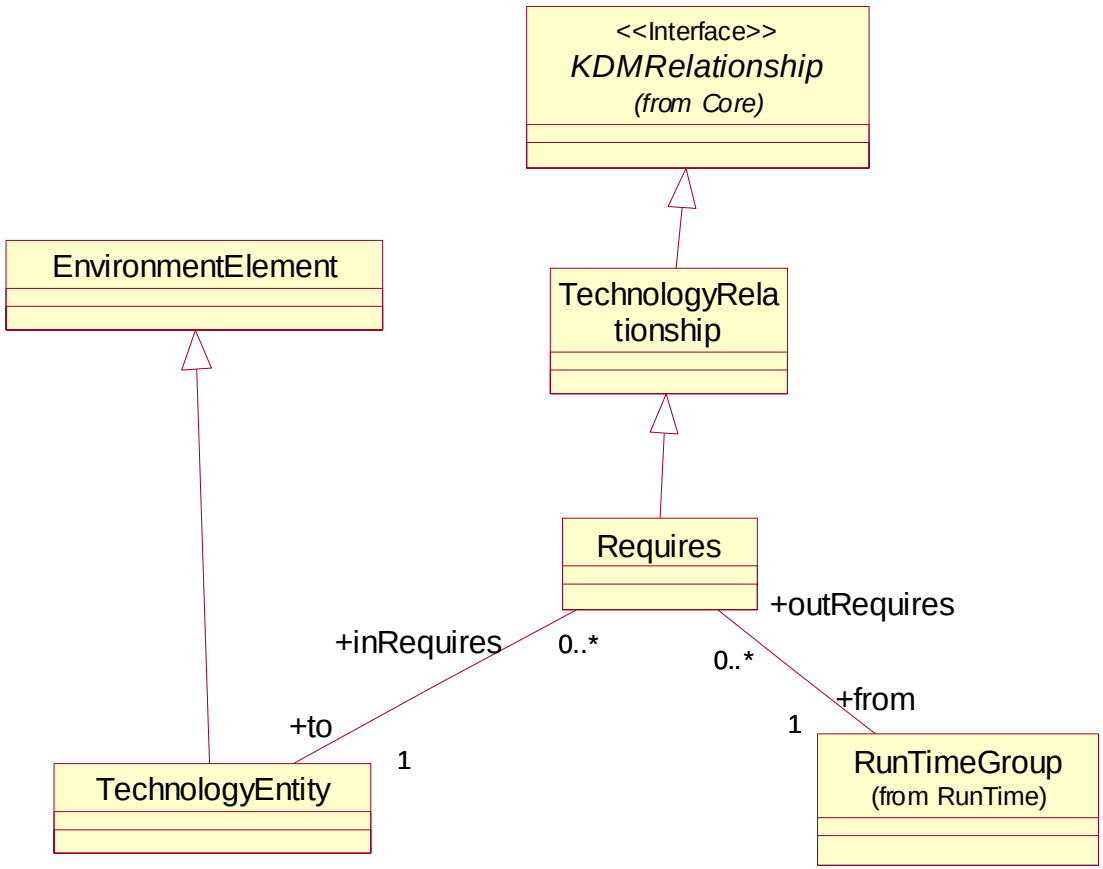
- EnvironmentEntity represents “things” in the operational environment of the system
- Relationships are derived from UI

Environment Package - services



- 3rd party components and services

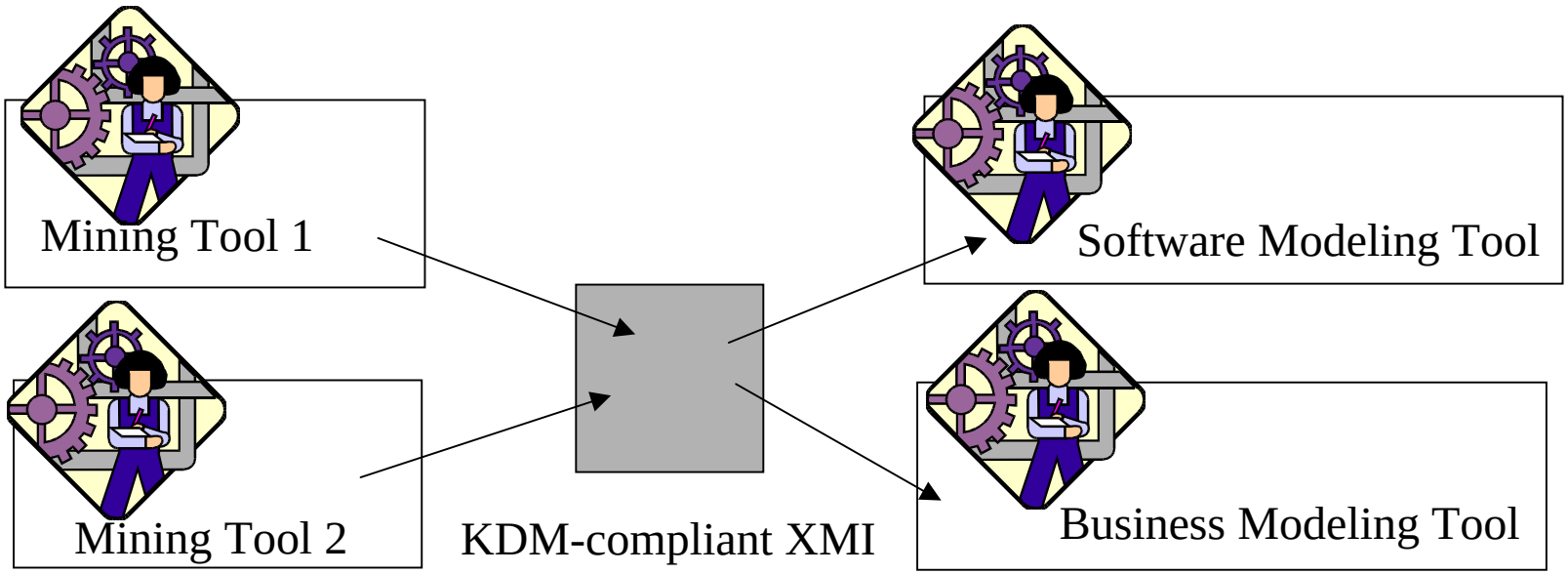
Environment Package - technology



- 3rd party implementations of platform element

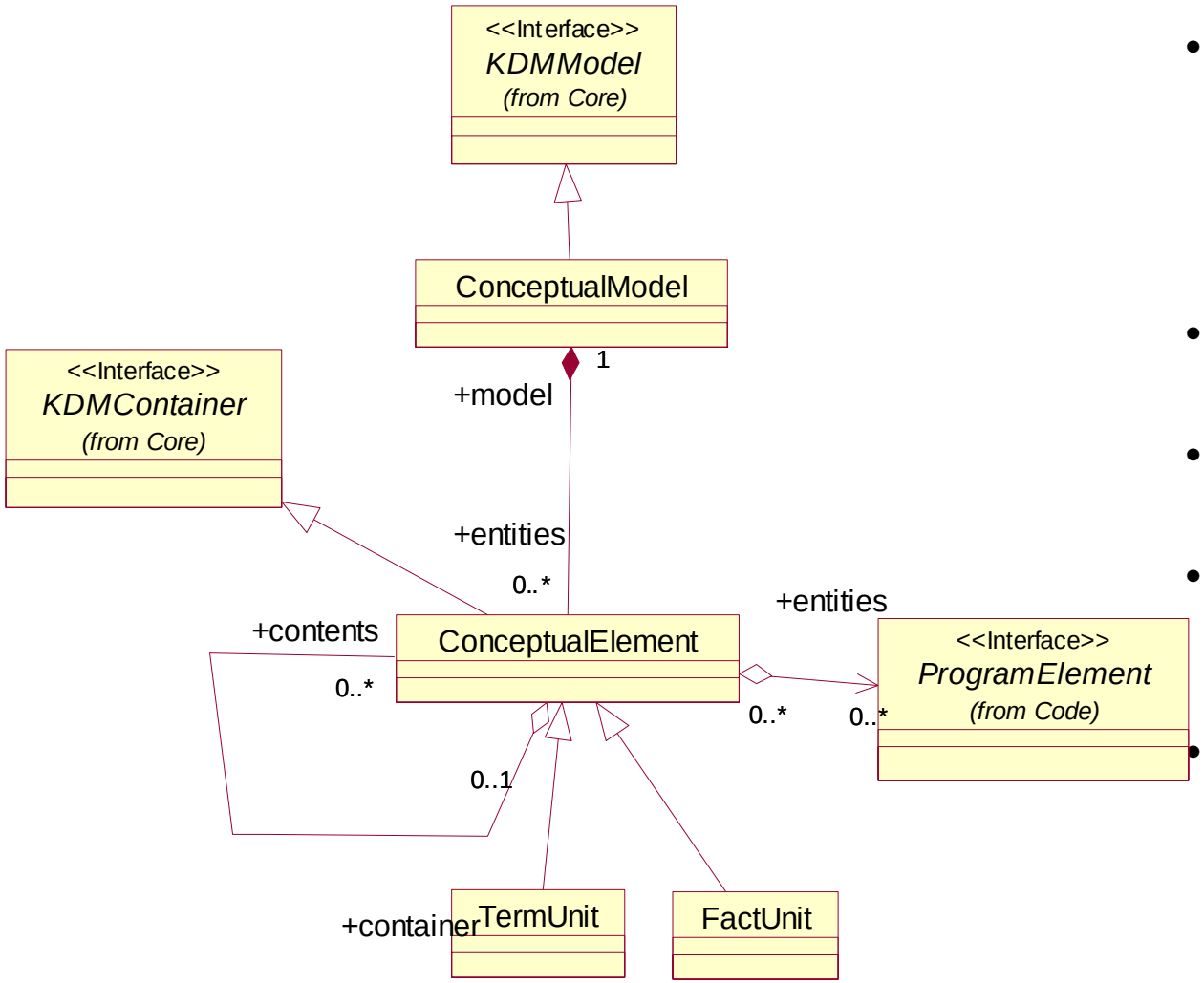
Conceptual and Behavior Packages

Interchange between Mining and Modeling Tools



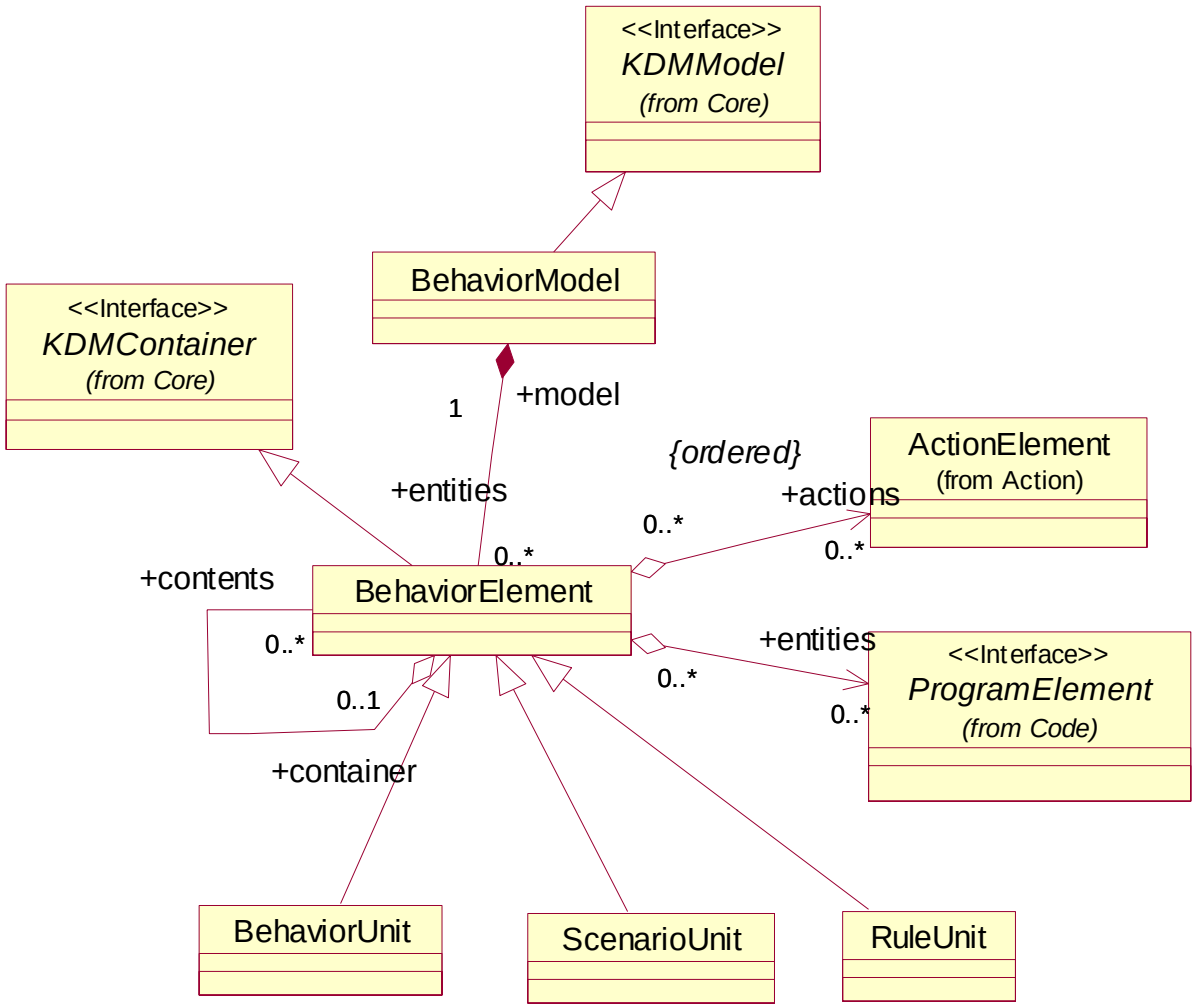
- **Integration of Modeling Tools with multiple Code Mining Tools**
- **Preservation of IP in a standard based repository**
- **Knowledge Mining and Abstraction to the required model abstraction level (e.g. Business Model)**
- **Enabling Forward Engineering Tools with discovered knowledge**

Conceptual Package



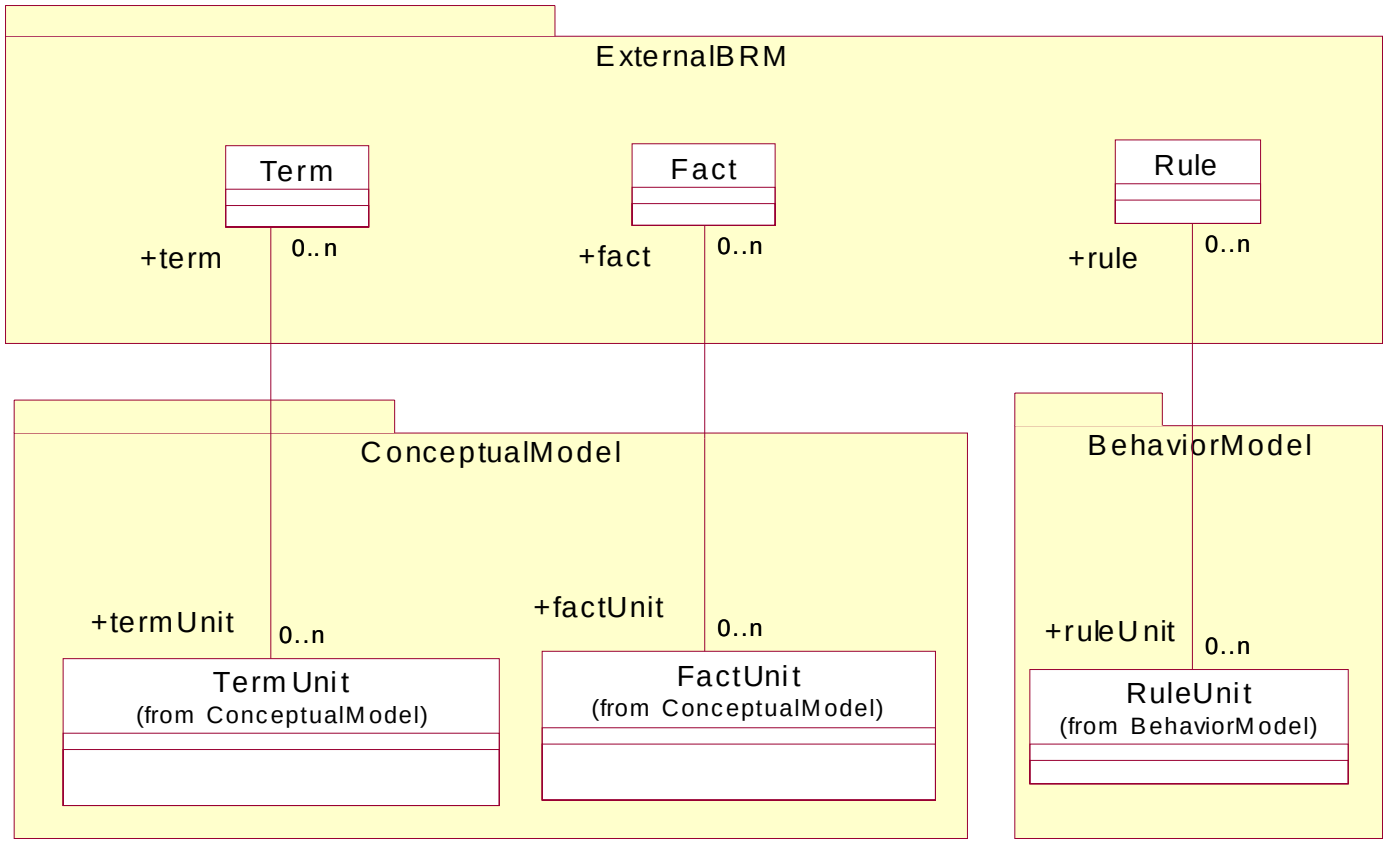
- ConceptualEntity represents a concept which is significant outside of the code in some external model
- ConceptulEntity is a set of CodeEntities
- ConceptualEntity allows composition
- Relationships are derived from CodeEntities
- Currently, there are 2 ConceptualEntities: TermUnit and FactUnit

Behavior Package



- BehaviorEntity represents behavior significant outside of the code in some external model
- BehaviorEntity is a directed graph of Actions
- BehaviorEntity allows composition
- Relationships are derived from Actions
- Currently there are 3 BehaviorEntities: BehaviorUnit, ScenarioUnit and RuleUnit

Mapping of External BRM to KDM



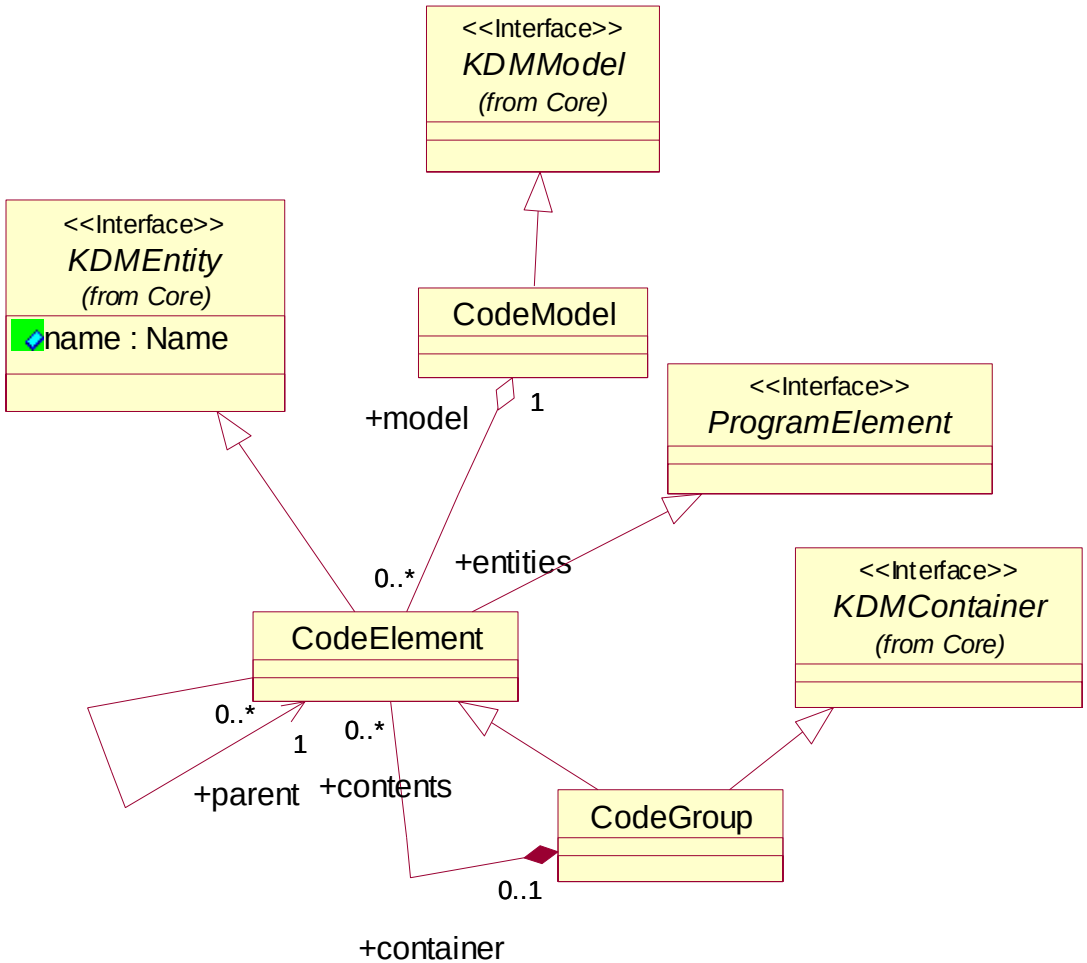
- KDM conceptual and behavior entities mapped to External BRM entities using many to many relationships
- KDM entities serve as implementation of BRM entities.

Code and Action packages

Code and Action Packages

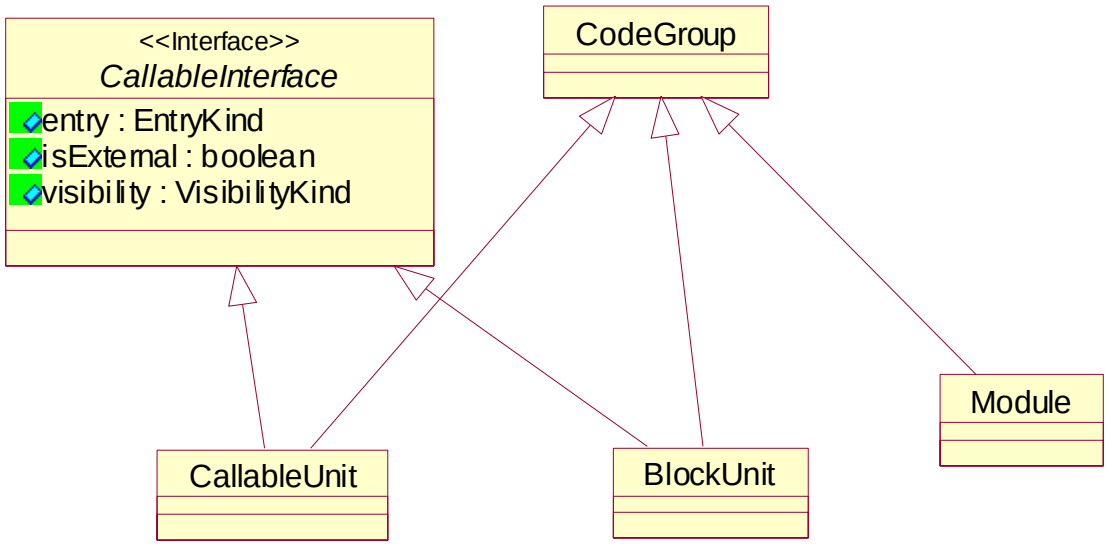
- Code Package:
  - Capture system artifacts that represents a block of instruction statements. For example a function in C or a paragraph in Cobol.
  - Code Package links to data package.
- Type Package:
  - Captures system artifacts that represent types
- Action Package:
  - Represents language independent executable statements behavior. For example Call, Control flow, IO.

Code package - main



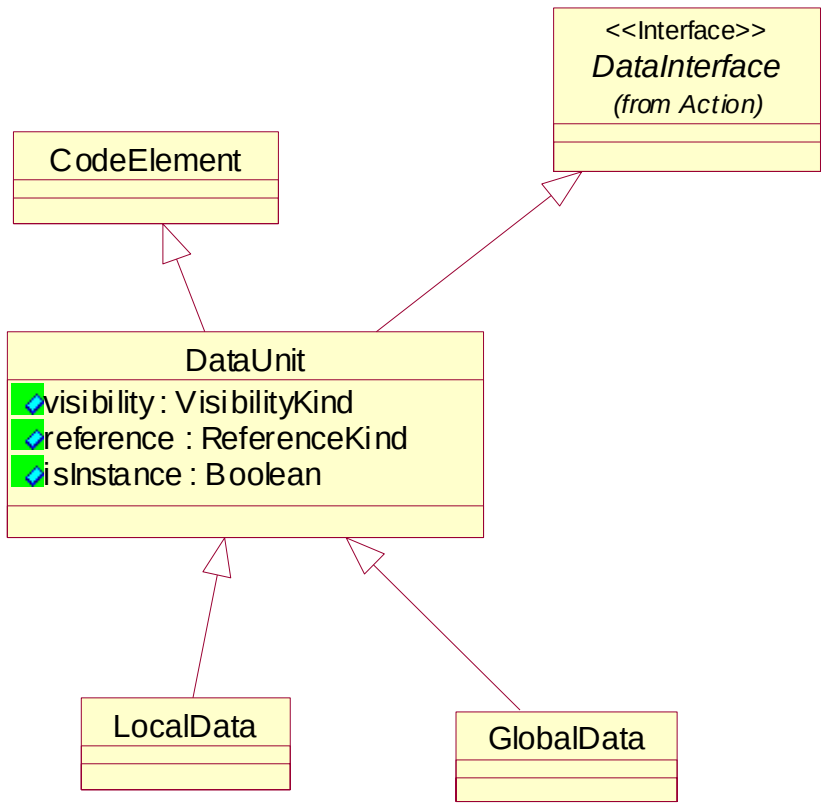
- Module is the container for code, data and actions
- CodeGroup allows composition

Code package – Code Elements



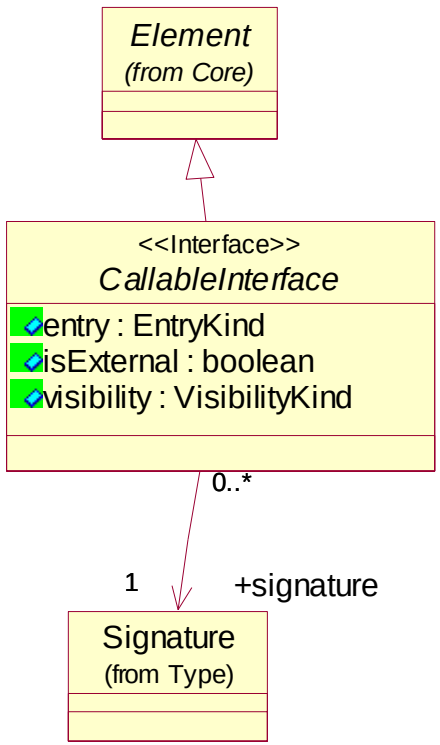
- CodeElements are implementation “things” in source files:
  - CallableUnit
  - ClassUnit
  - DataUnit
  - MacroUnit
  - PrototypeUnit
  - TemplateUnit
- This models defines some specific relations
- Data type elements are defined in a separate package

Code package – Code Entities



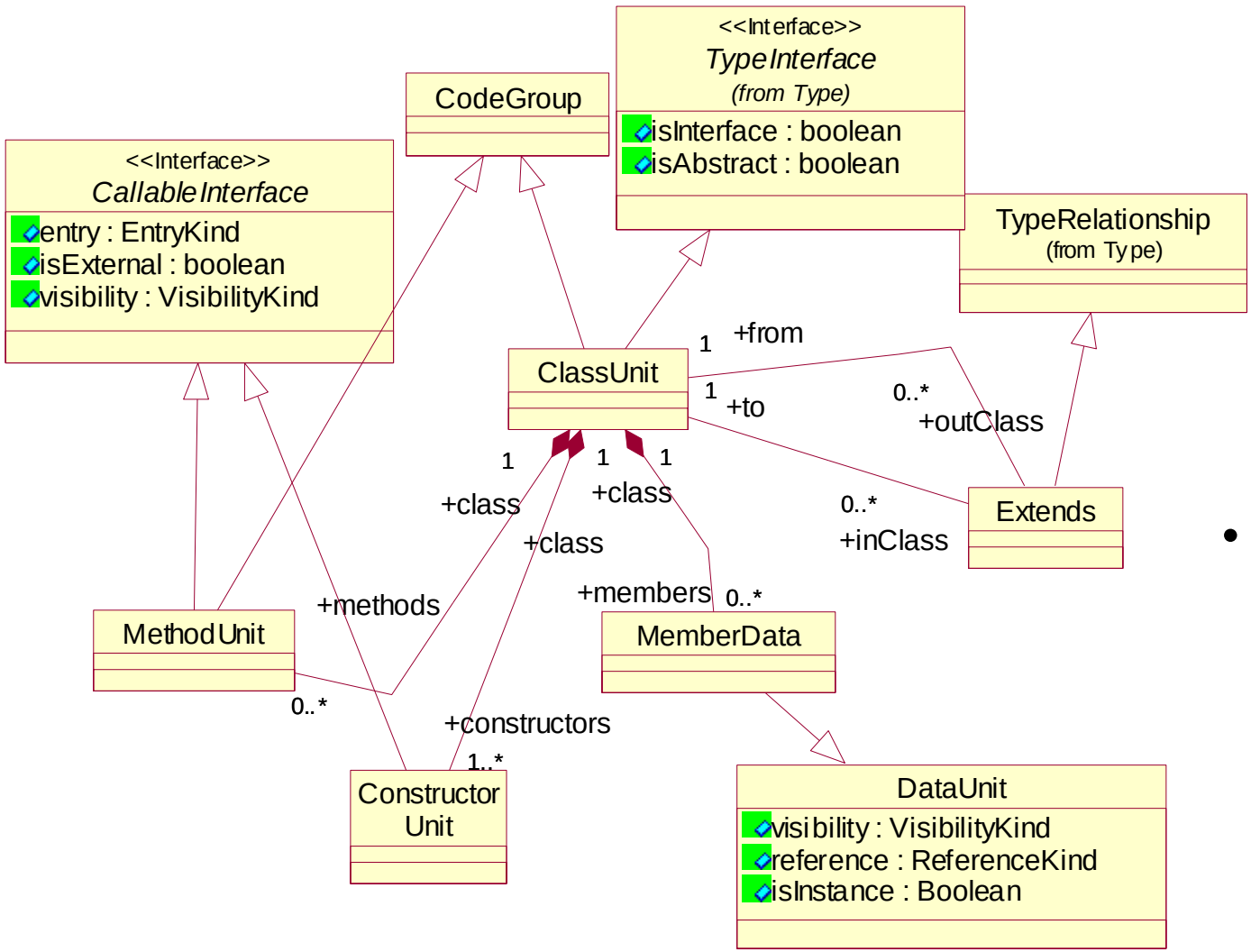
- CodeEntities are implementation “things” in source files:
  - CallableUnit
  - ClassUnit
  - DataUnit
  - MacroUnit
  - DeclarationUnit
  - DataTypeUnit
- This models defines some specific relations

Code package – callable unit



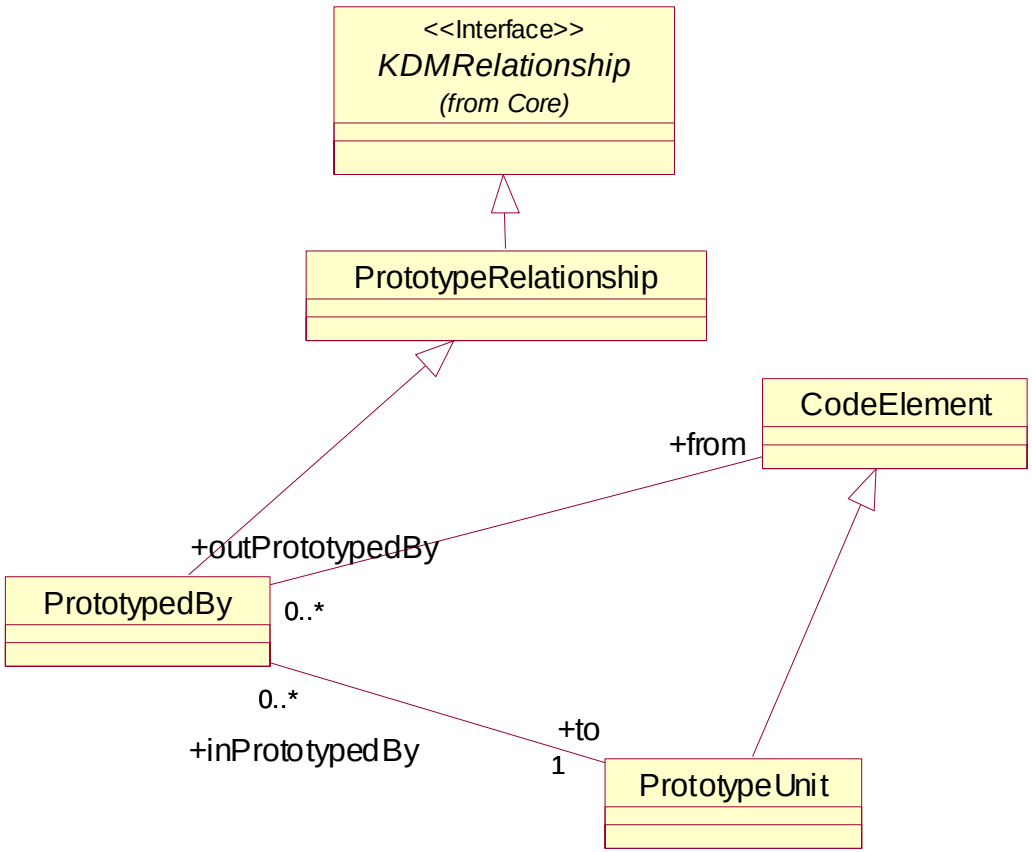
- CallableUnit contains Signature (including ParameterData)

Code package – class Unit



- ClassUnit contains MethodData and MemberData
- Constructors are represented separately from methods
- ClassUnit can subclass other ClassUnits

Code package - prototypes

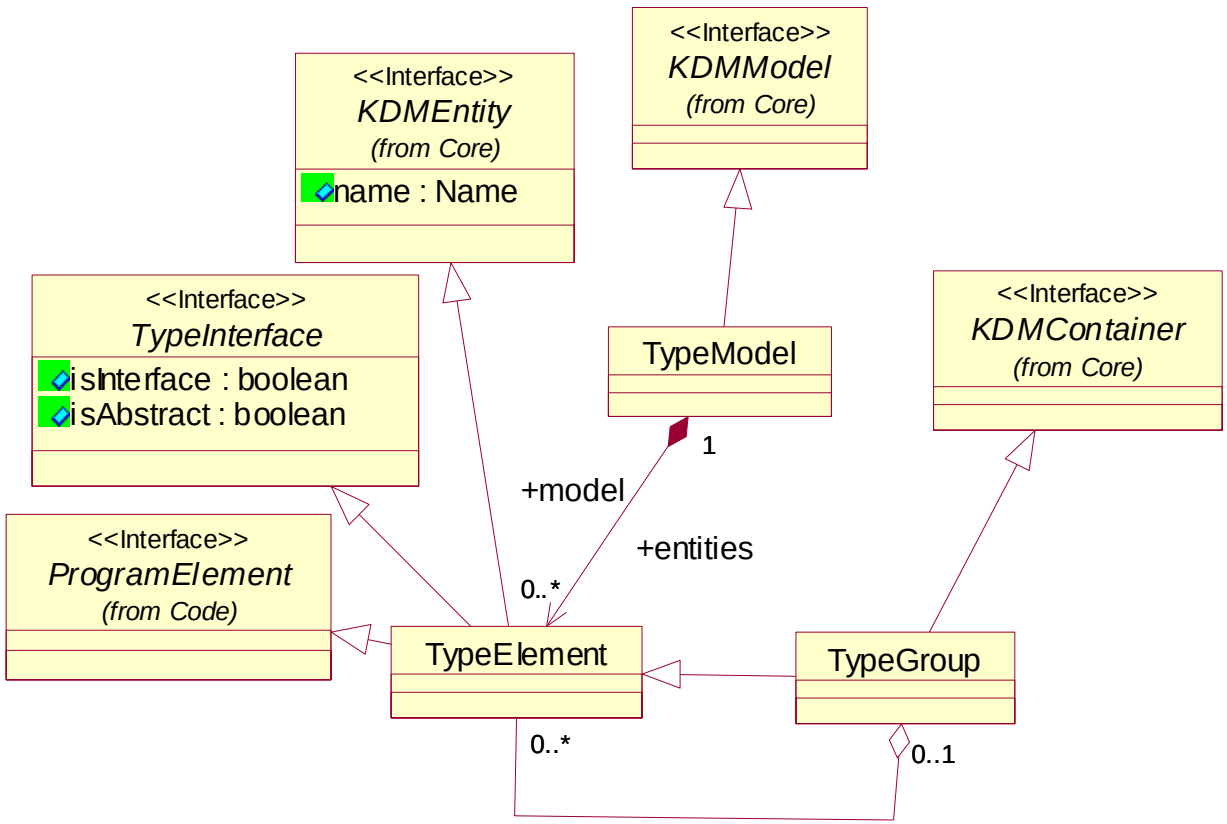


- PrototypeUnit defines some specific relations

Code package – other elements

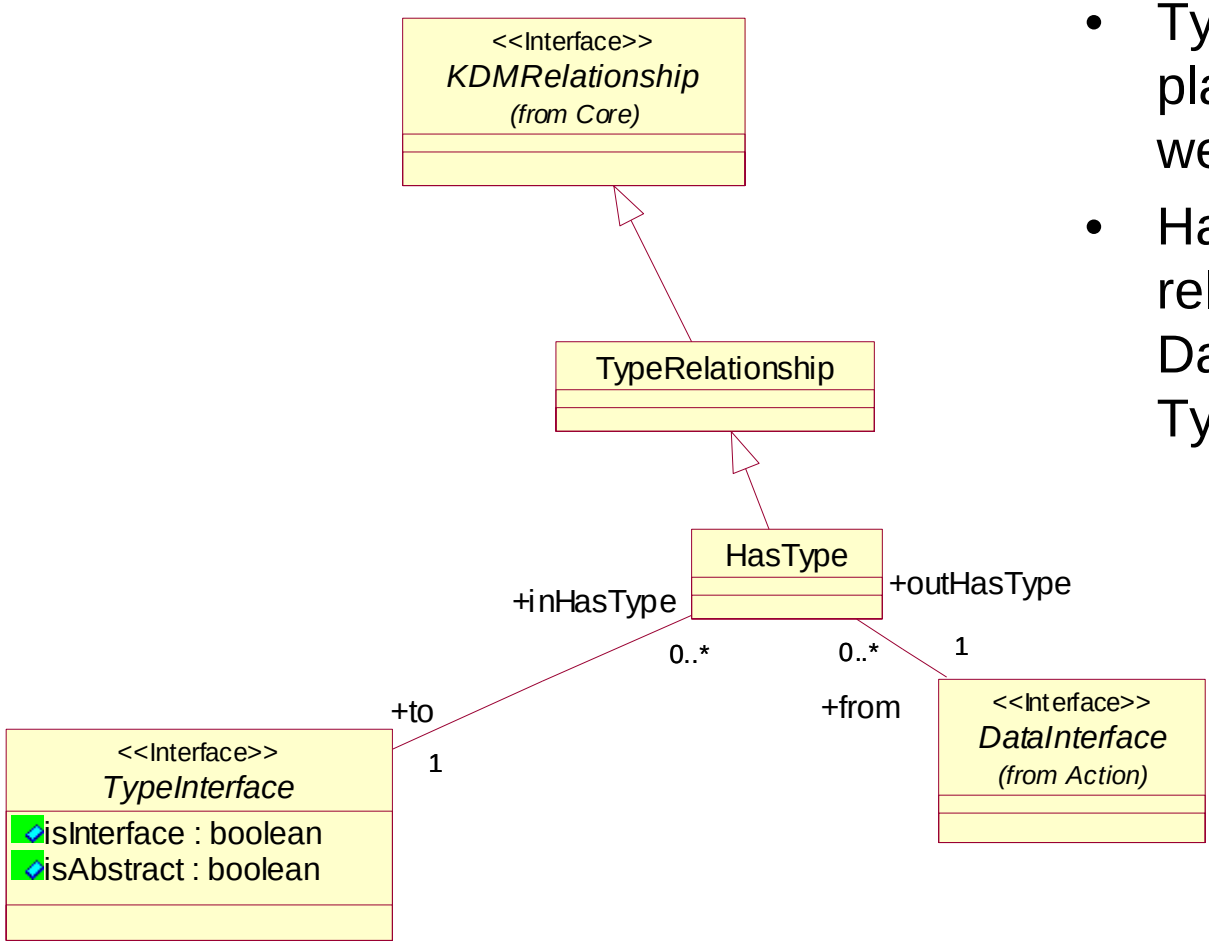
- MacroUnit
- TemplateUnit, TemplateInstance and TemplateParameter

Type package - main



- Type package represents program element that are related to types

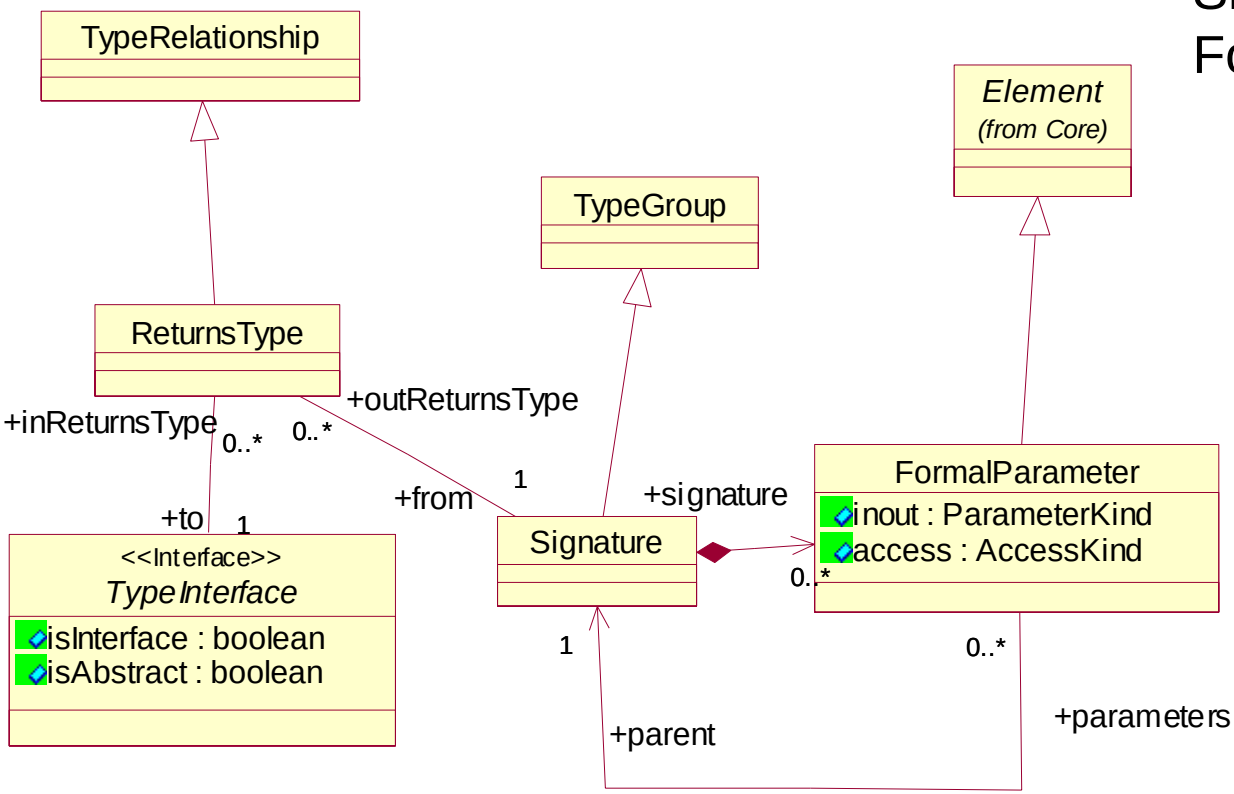
Type package – Type Relations



- Type Relation is a placeholder for light-weight extensions
- HasType is a relationship between a DataInterface and a TypeInterface

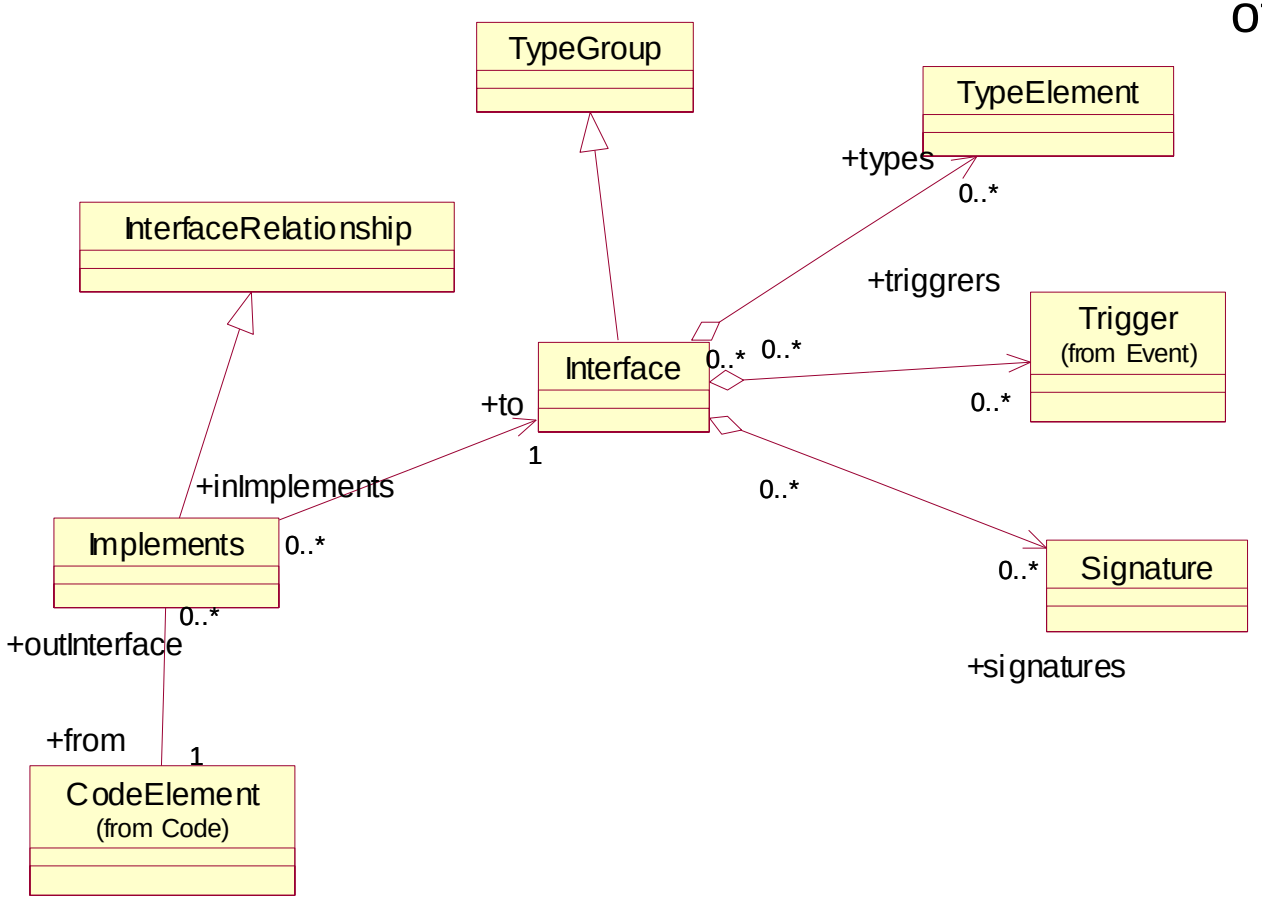
Type package - Signature

- Signature contains Formal Parameters

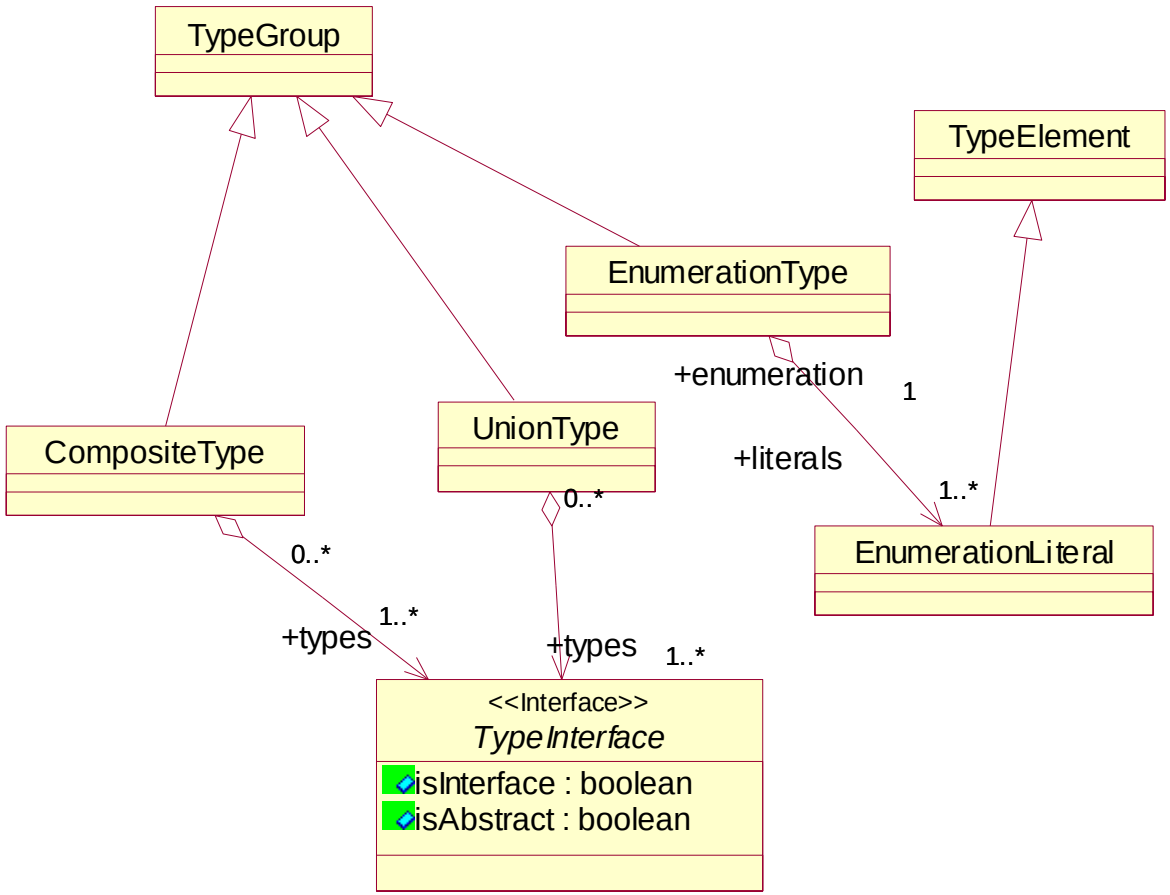


Type package - Interface

- Interface is a collection of types



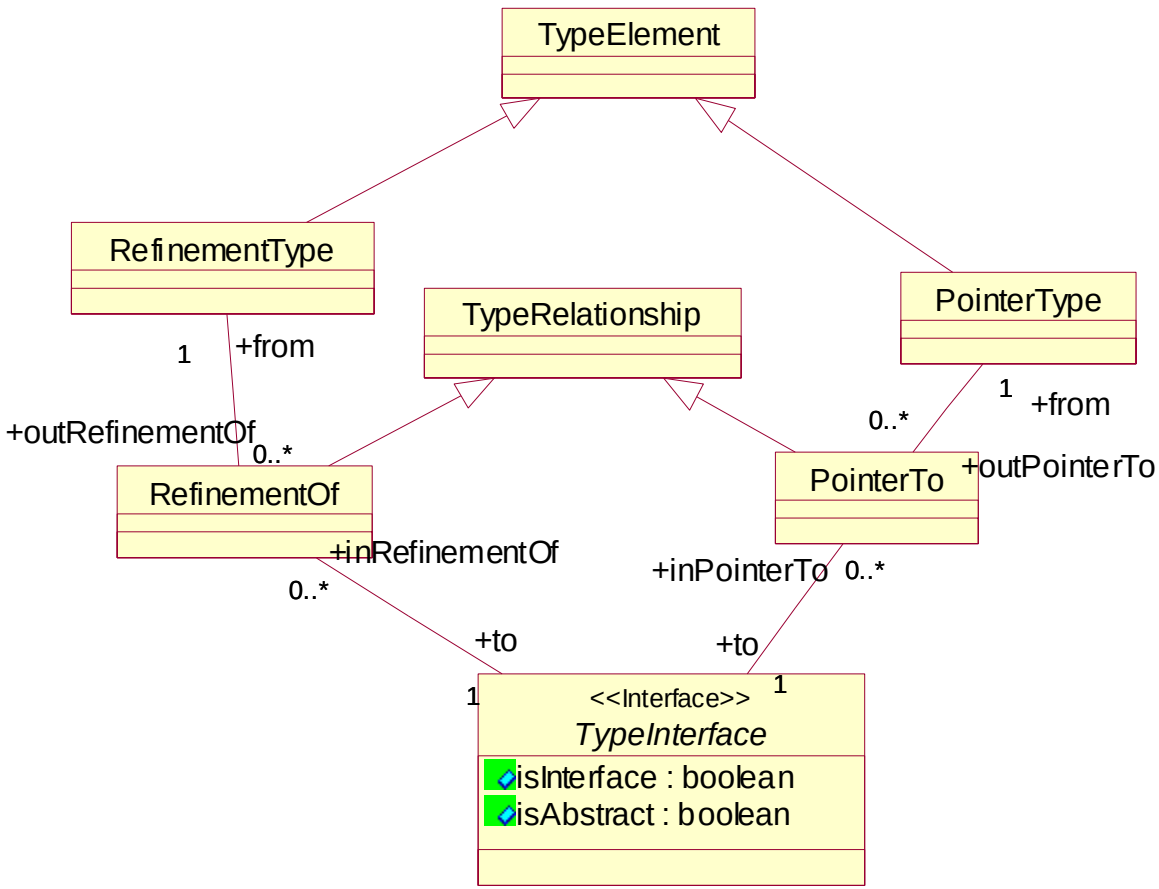
Type package – Composite Type Elements



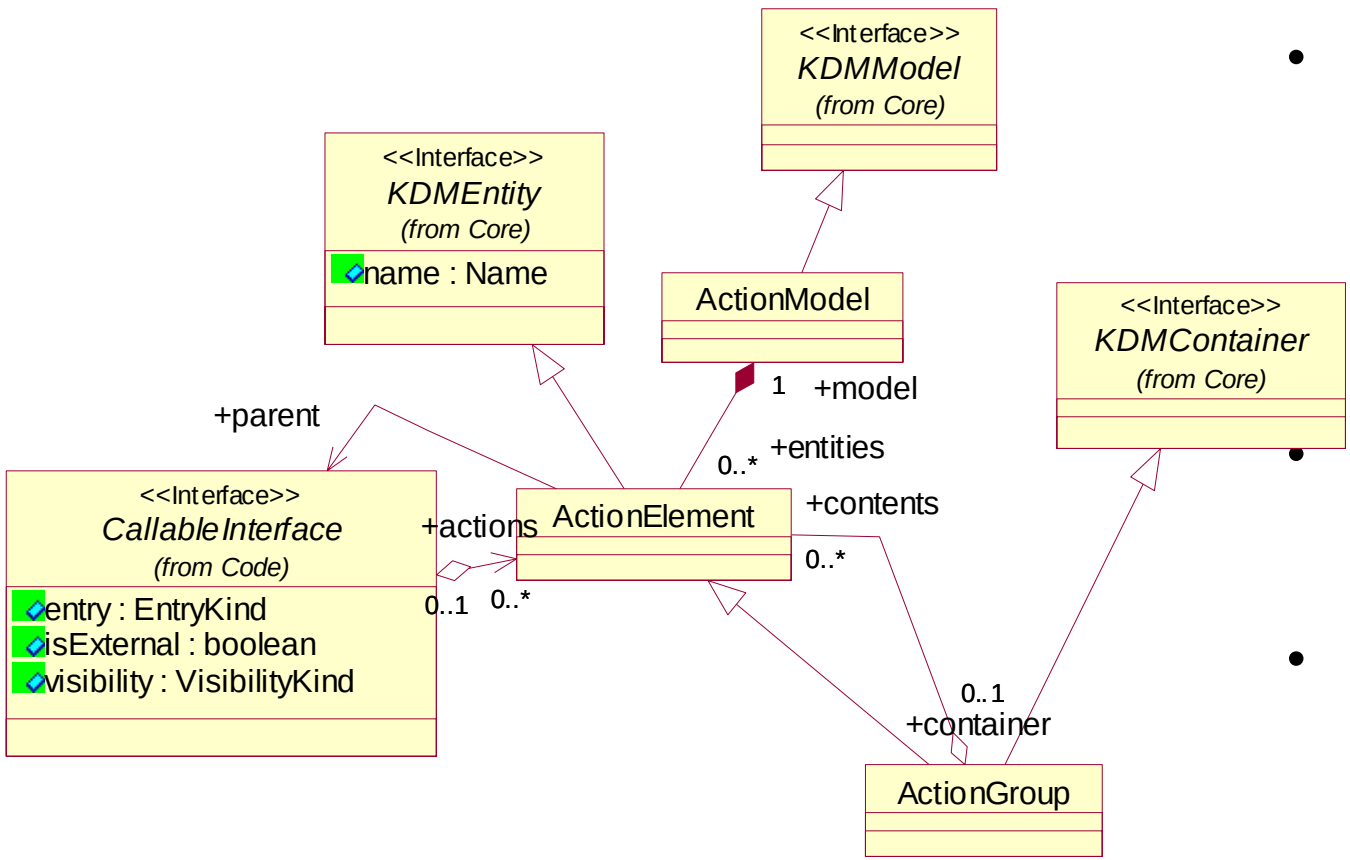
- Composite Type Elements are containers

Type package – Type Elements

- Non-composite types

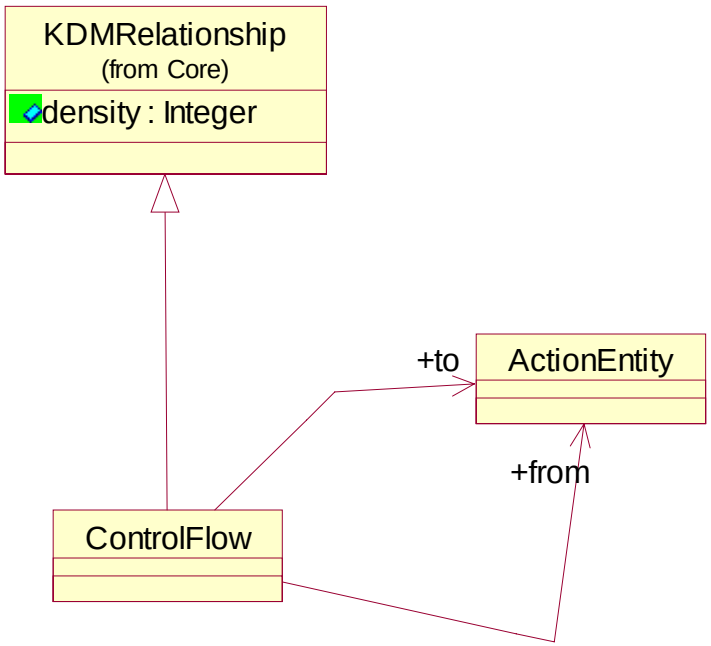


Action package – Action Entities



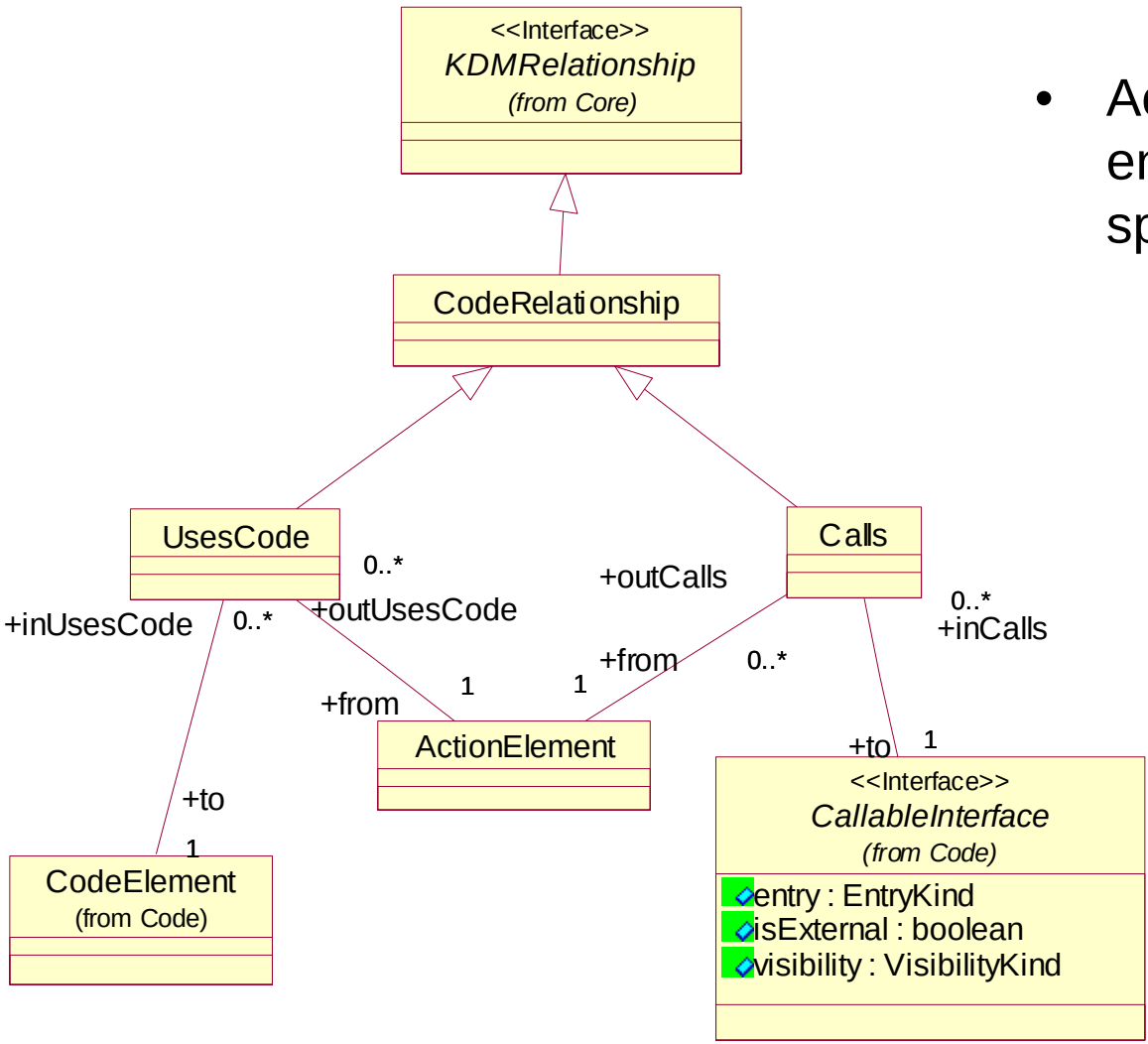
- ActionEntity represents behavior “things”, like statements, features, business rules, etc.
- ActionGroup allows composition
- ActionEntities are endpoints for some specific relations

Action package – action flow



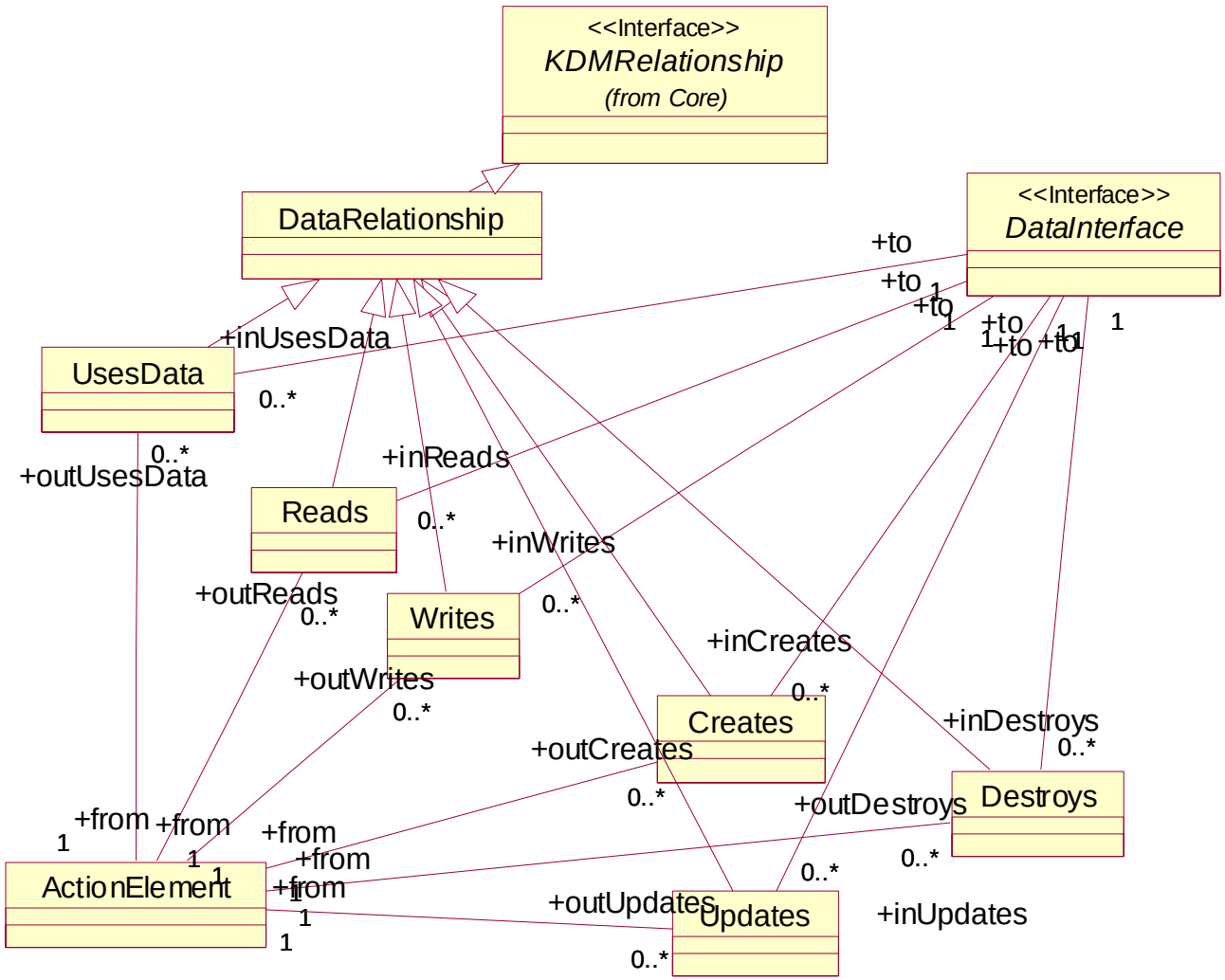
- There is ControlFlow relation between ActionEntities

Action package – Code relationships



- ActionEntities are endpoints for some specific relations

Action package – data relationships



- ActionEntities are endpoints for some specific relations to Data via DataInterface

Action package – other relationships

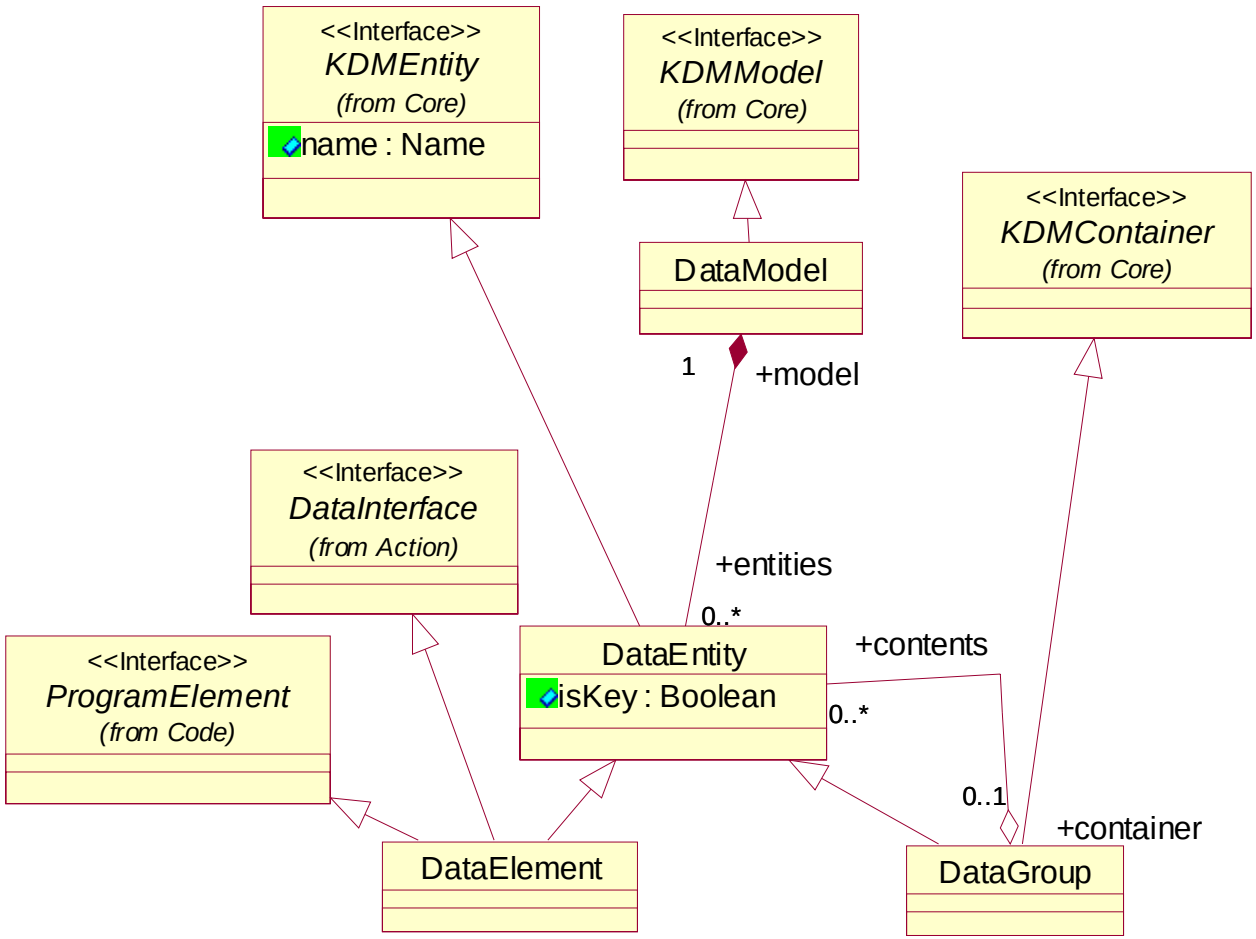
- Import relations
- Macro relations
- Prototype relations
- Type relations

Data package

Data Package

- Data Package:
  - Defines types for elements
  - Represents system artifacts that define system data
  - Artifact representations include persistent and non-persistent data
- Data Package defines data elements and data groups along with links to persistent data representations and data models
- Data Package links to data definitions as expressed in the Code Package

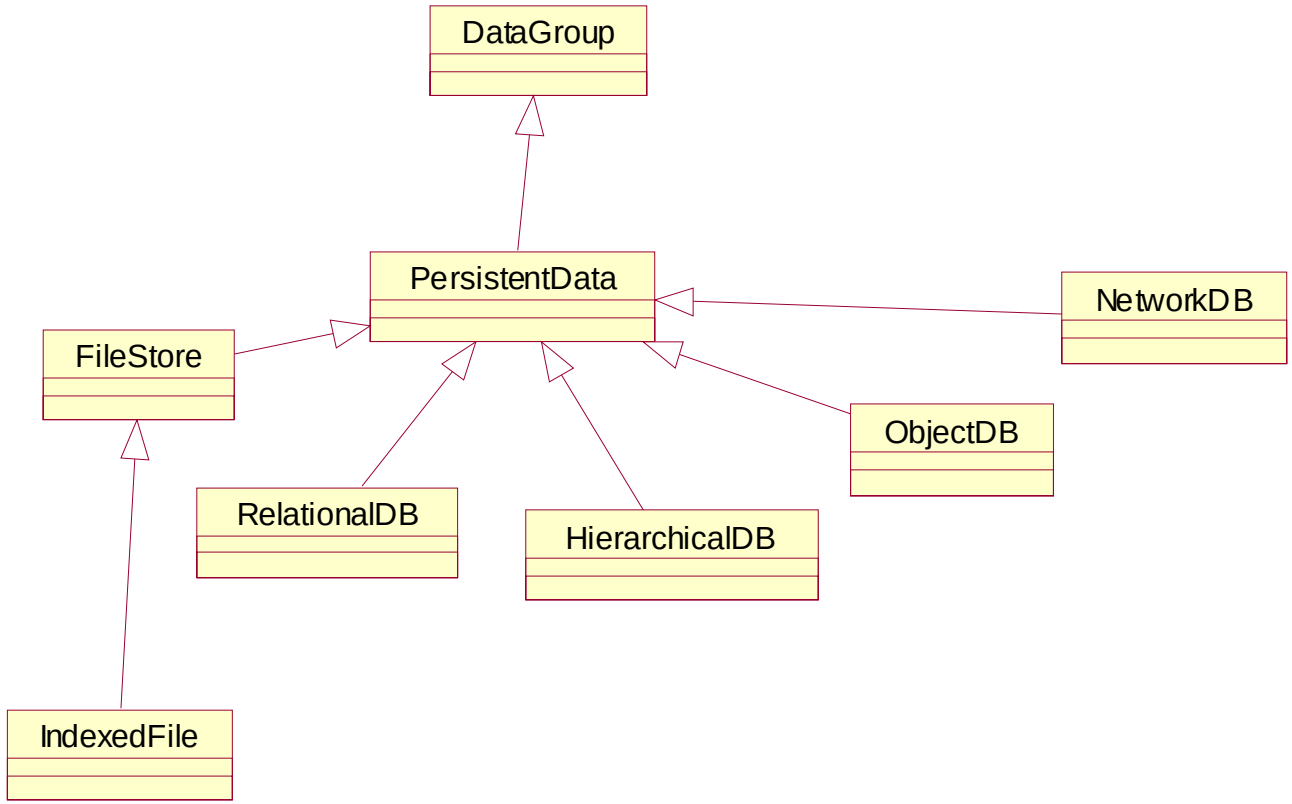
Data Package



Data Elements

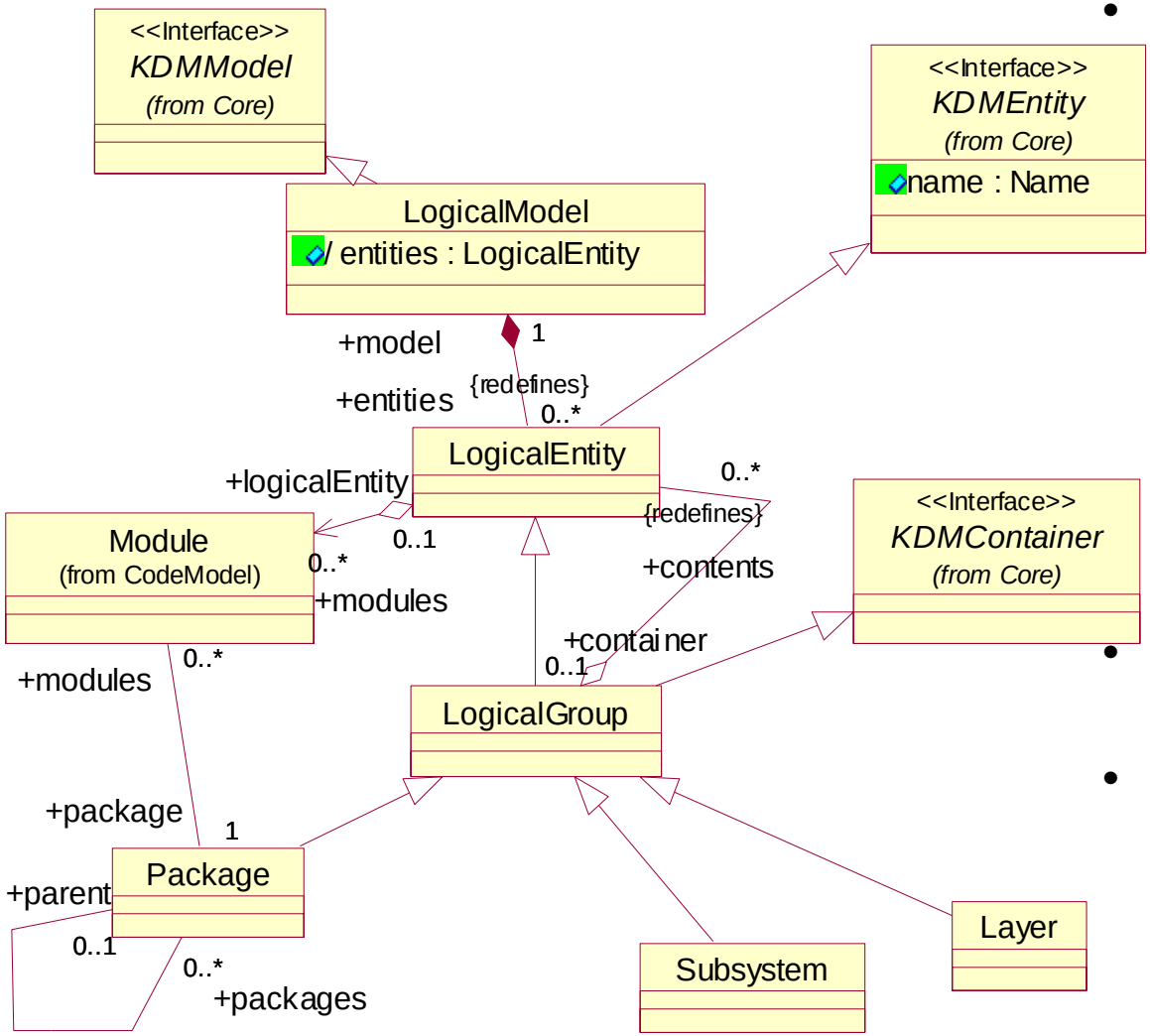
- Data Element: Named data field extracted from a system that may be a group or an individual element.
- Data Entity: Logical data element that represents a relational entity in a data model
- Data Group: A collection of data elements, typically derived from existing system record, table, segment or similar grouping structures.
- A Data Element may be tied directly to Persistent Data. Persistent data may be represented in a data model.

Data Package – persistent data



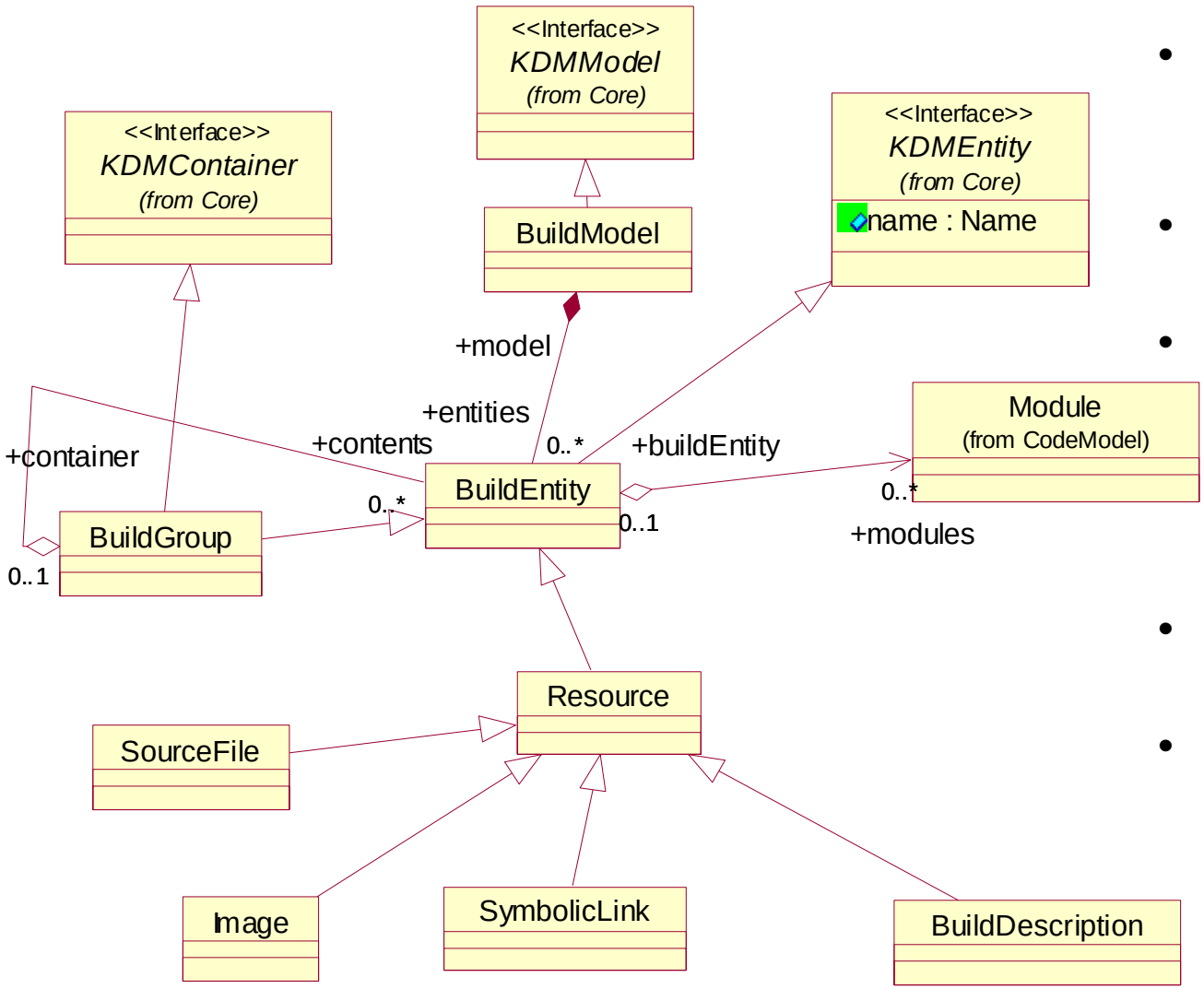
Logical, RunTime and Build Packages

Logical Package



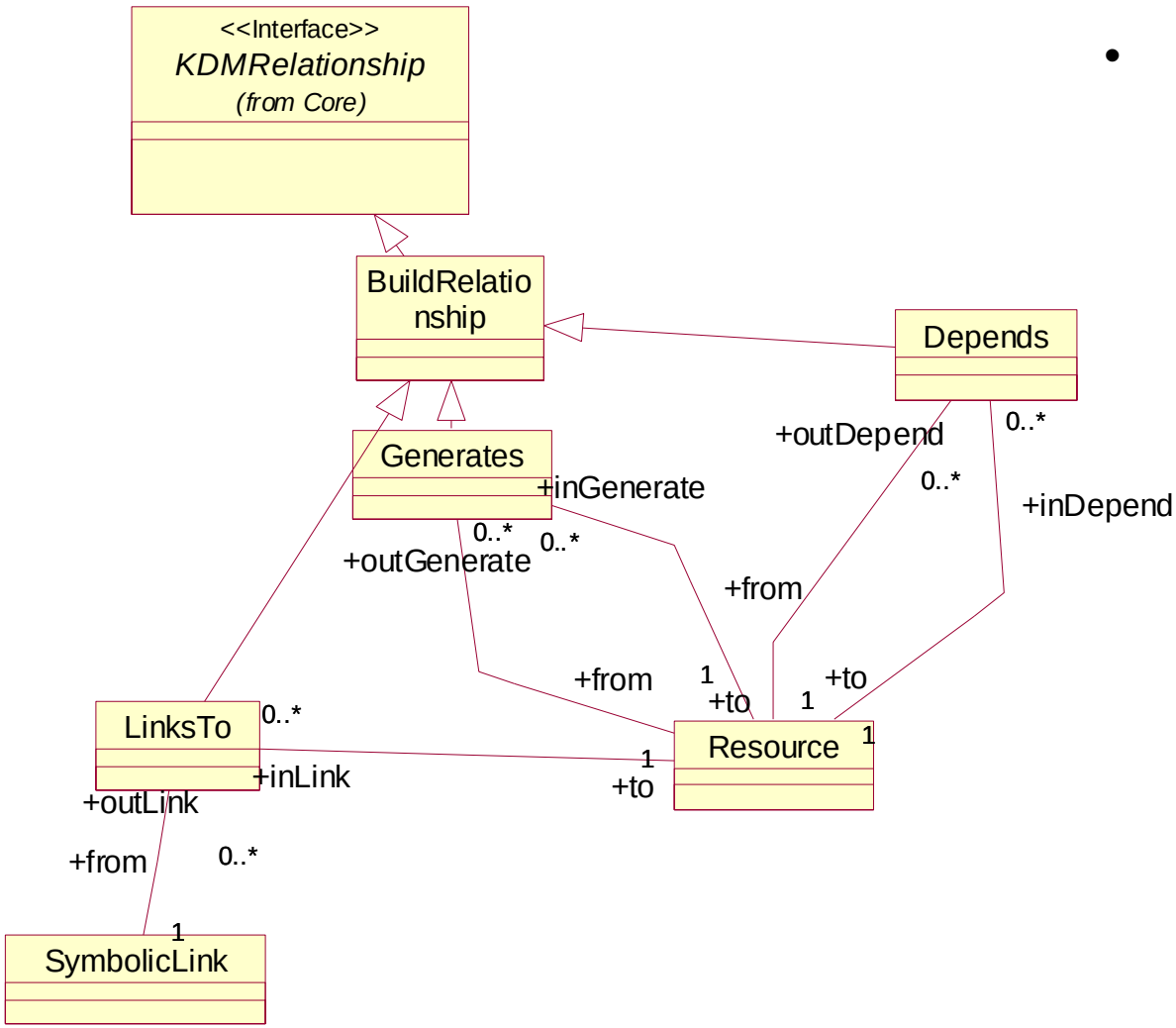
- Logical package describes components of existing software
- It represents structural "elements":
  - Package
  - Subsystem
  - Layer
  - LogicalGroup which allows composition
- Module is the container for code, data and actions
- Relationships between components are derived from Modules

Build Package - main



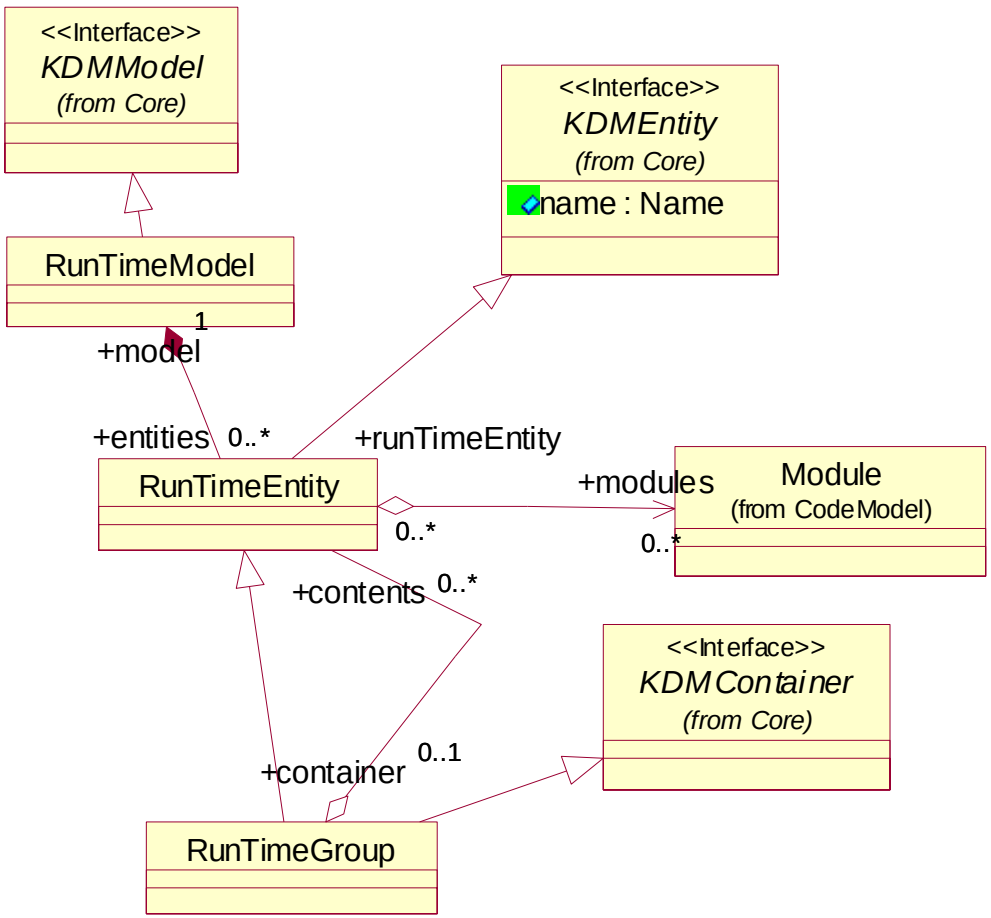
- Build package represents physical “things”:
- BuildGroup allows composition
- BuildComponent contains Modules
- Module is the container for code, data and actions
- BuildComponent contains Files
- This model defines specific relations

Build Package – build relations



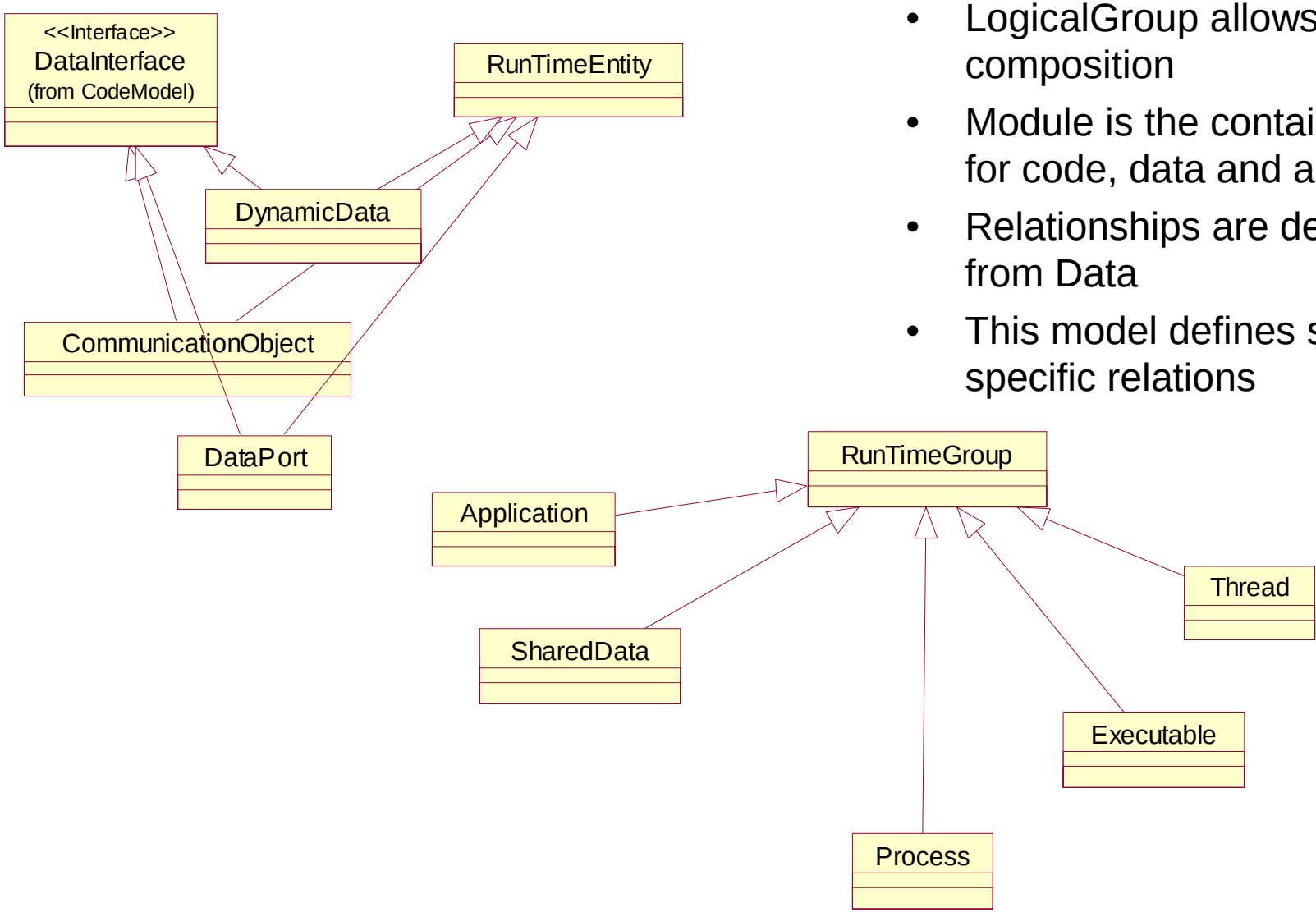
- Specific build relations:
  - File generates file
  - File imports file
  - Files depends on file
  - SymbolicLink points to File

RunTime Package



- RunTime package represents common runtime platform aspects:
  - Platform Resource
  - Application Component
  - Platform Service
  - Platform Binding
- It addresses such things as:
  - Application
  - Executable
  - Process
  - Thread
- RunTime model also represents inter-component communication:
  - Communication object
  - Shared data

RunTime model - Groups



- LogicalGroup allows composition
- Module is the container for code, data and actions
- Relationships are derived from Data
- This model defines some specific relations

UI package

User Interface Package

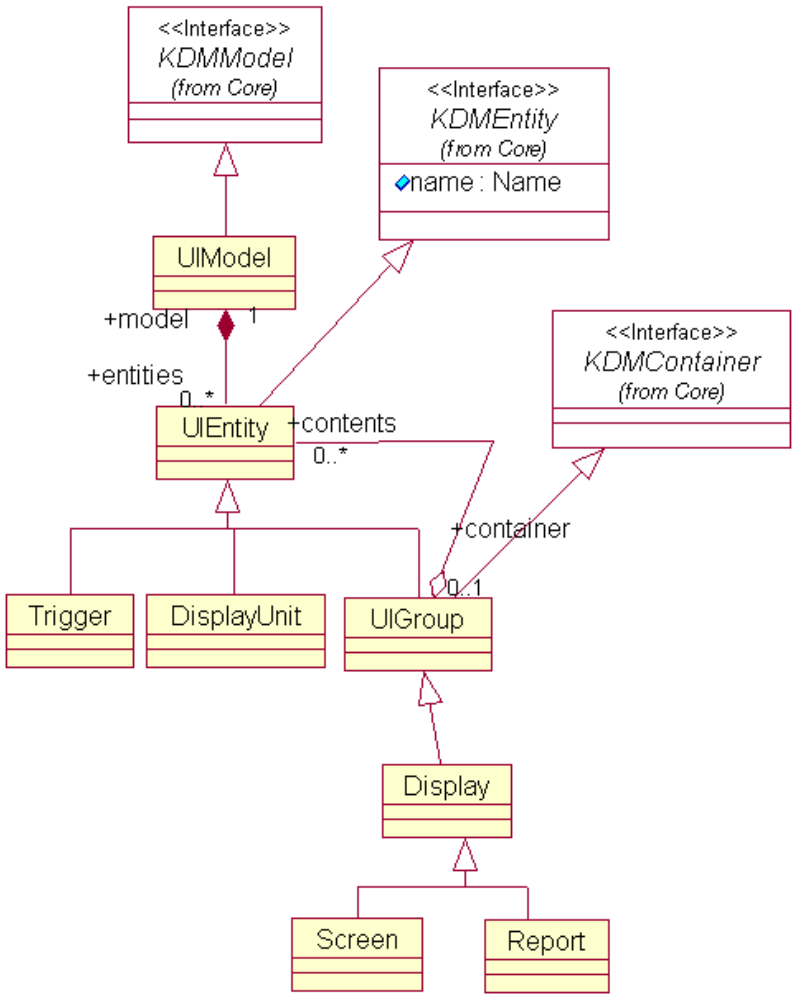
- Overview
  - The UI Package takes a broad view of visual (elements, views) and behavioral aspects (events) of the user interface of a software application

- Dependencies (other than KDM Core)
  - Resources defined in the BuildModel may be used in a view
  - DataInterfaces from the CodeModel may be presented in a view
  - CodeInterfaces from the CodeModel may be associated with a UI element and with UI events

UI Package diagrams

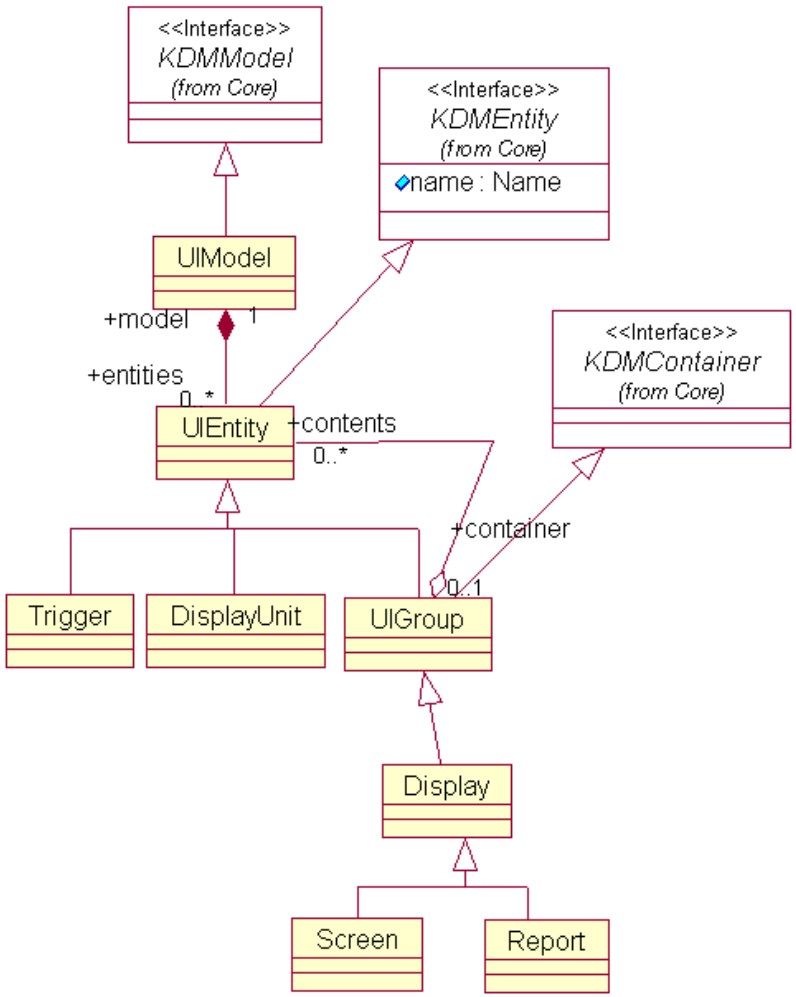
- UI core entities and relations
- UI layout
- UI behavior
- UI events

UI core entities and relations



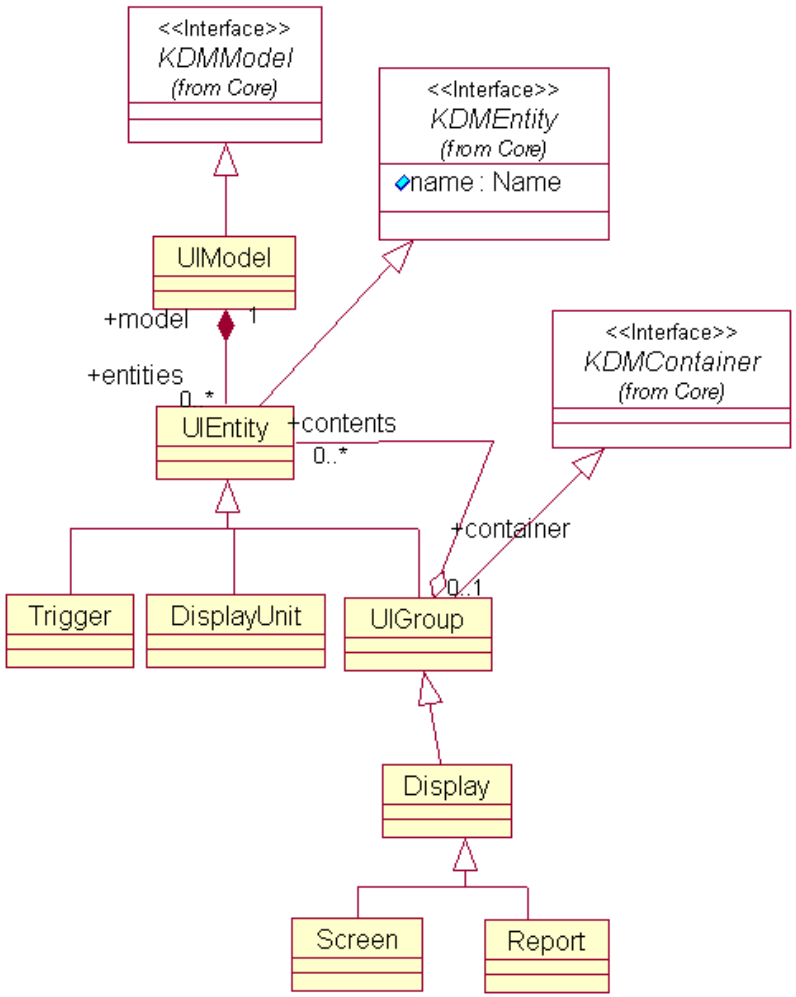
- From KDM Core
  - UIModel from KDMModel
  - UIEntity from KDMEntity
  - UIGroup from KDMContainer

UI core entities and relations (cont.)



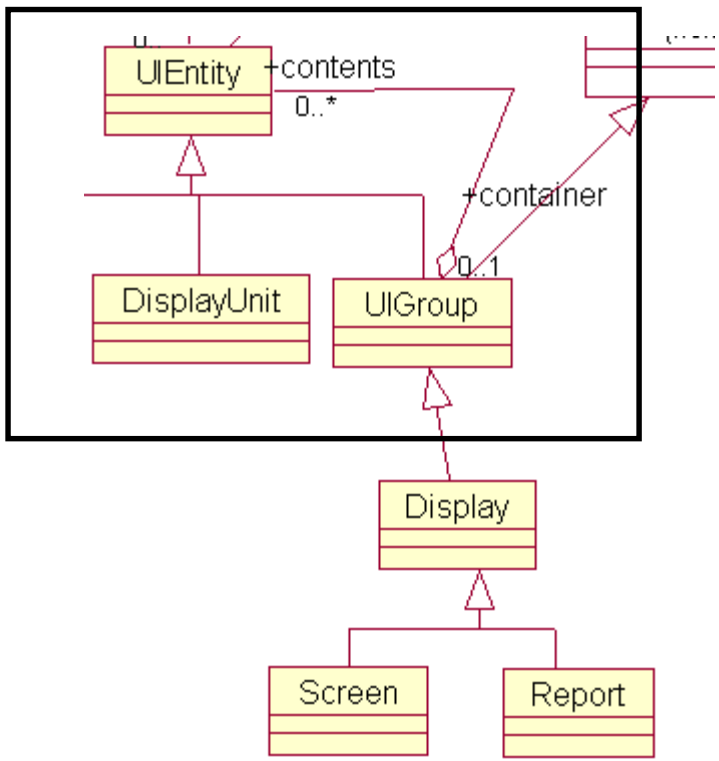
- Trigger –events (e.g. key, mouse, timer) linked to display or CodeInterfaces
- DisplayUnit – single UI entity (e.g. a control on a form)
  - May be a resource from BuildModel
  - May be a DataInterface from CodeModel

UI core entities and relations (cont.)



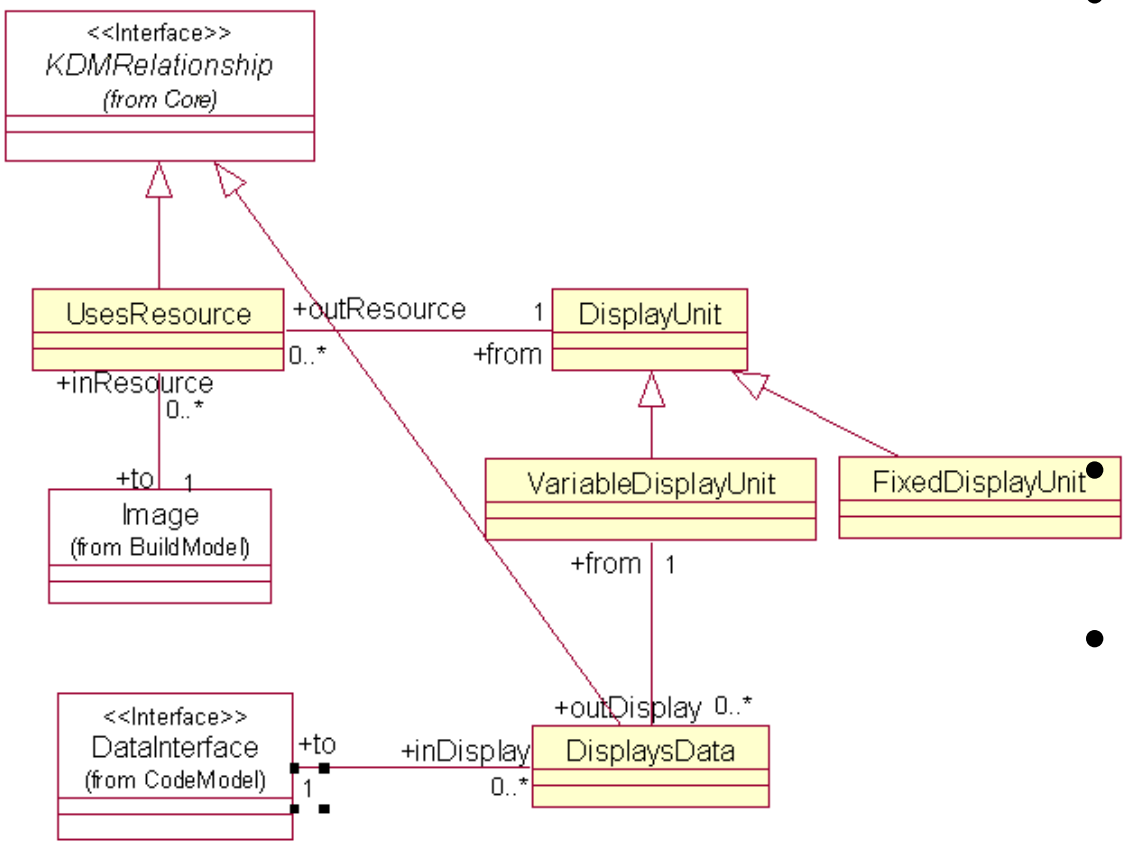
- UIGroup – composition of other UI entities
- Display – a compound unit of display (e.g. a Web page) that is composed of other display elements
  - Screens
  - Reports

UI layout



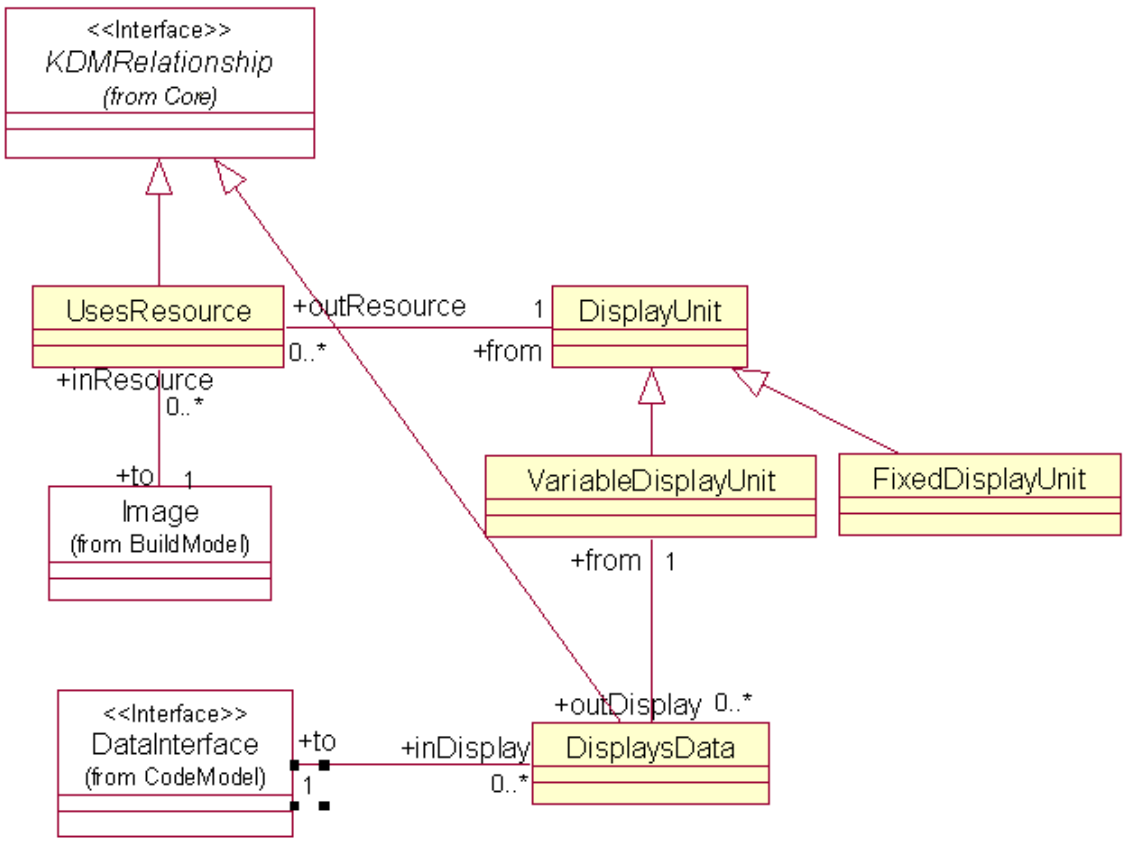
- One aspect of UI layout is captured in the UI core model just presented: the UIGroup is composed of instances of UIEntity, such as DisplayUnits

UI layout (cont.)



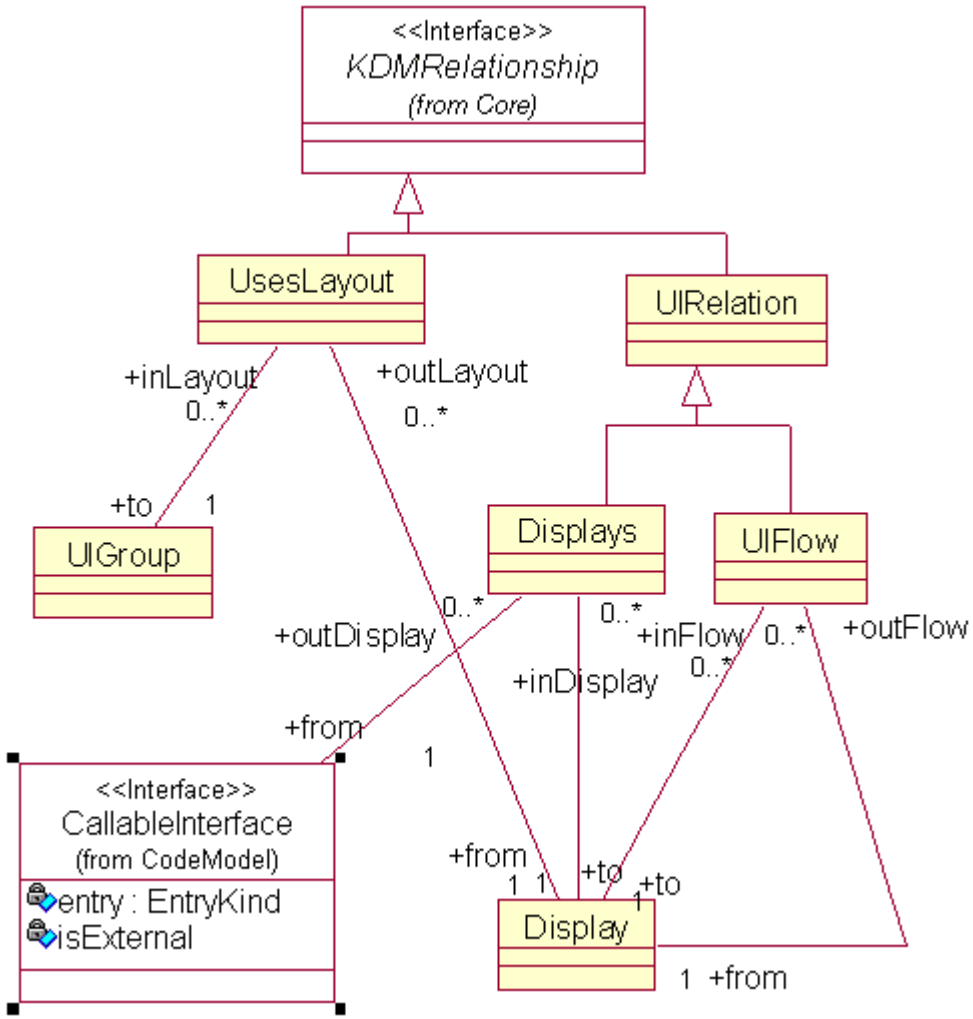
- These classes and relationships further define the linkage between DisplayUnits and the information they render
- A FixedDisplayUnit is static information
- A VariableDisplayUnit is dynamic information

UI layout (cont.)



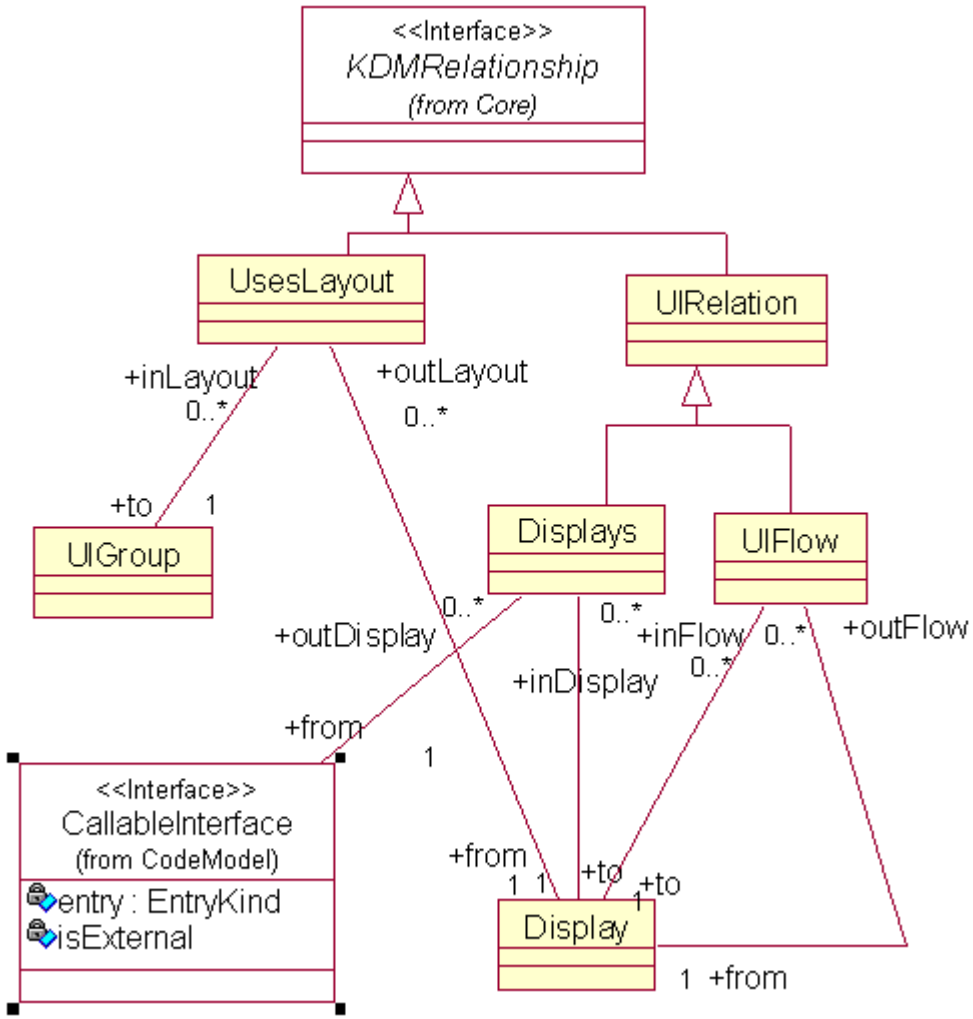
- UsesResource is the relationship class linking the UI model to the BuildModel for interface content (e.g. images)
- DisplayData is the relationship class linking the UI model to the CodeModel for data

UI behavior



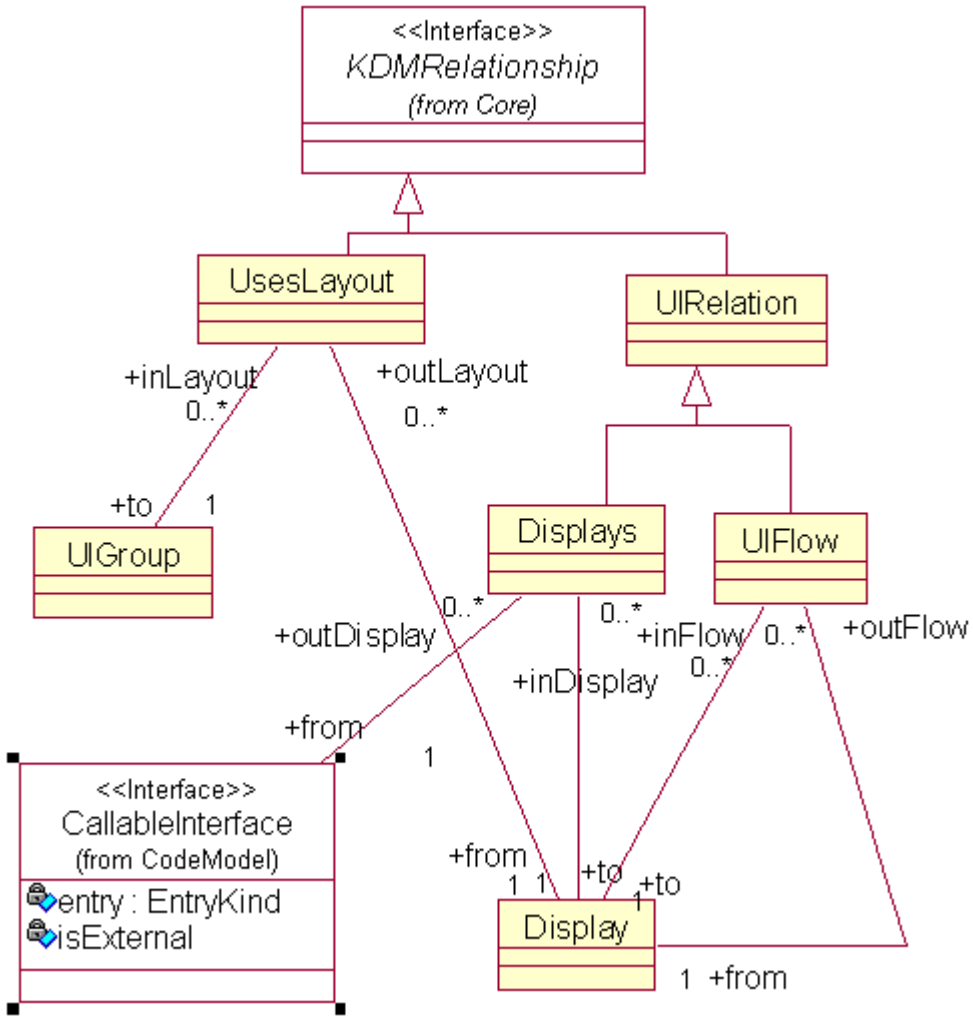
- The high-level behavior of the user interface is recorded as the flow from one Display instance to another, using the UIFlow relationship class

UI behavior (cont.)



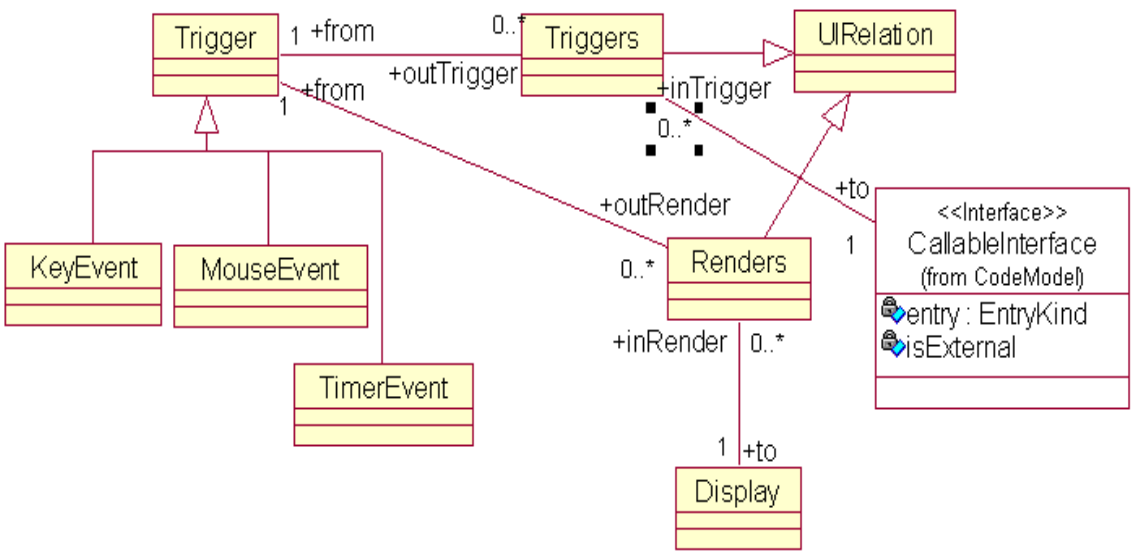
- The relationship between the software application software and the UI Display is recorded in the Displays relationship class.
- The CodeModel's CallableInterface activates an instance of Display.

UI behavior (cont.)



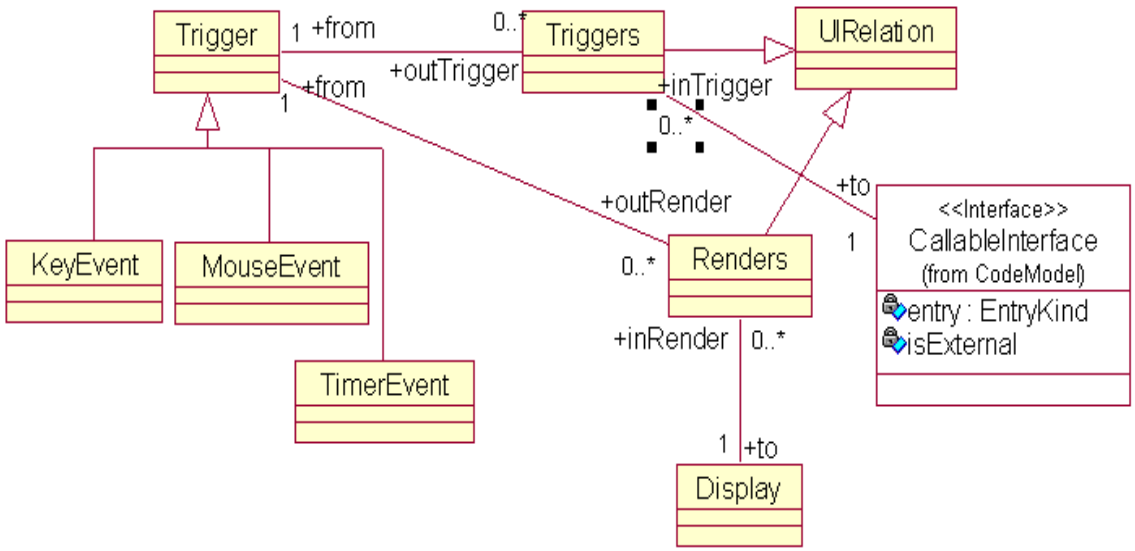
- Refresher:
  - Display is a compound unit of display (e.g. a Web page) that is composed of other display elements
- Separation of layout and content is modeled by the UsesLayout relationship class

UI events



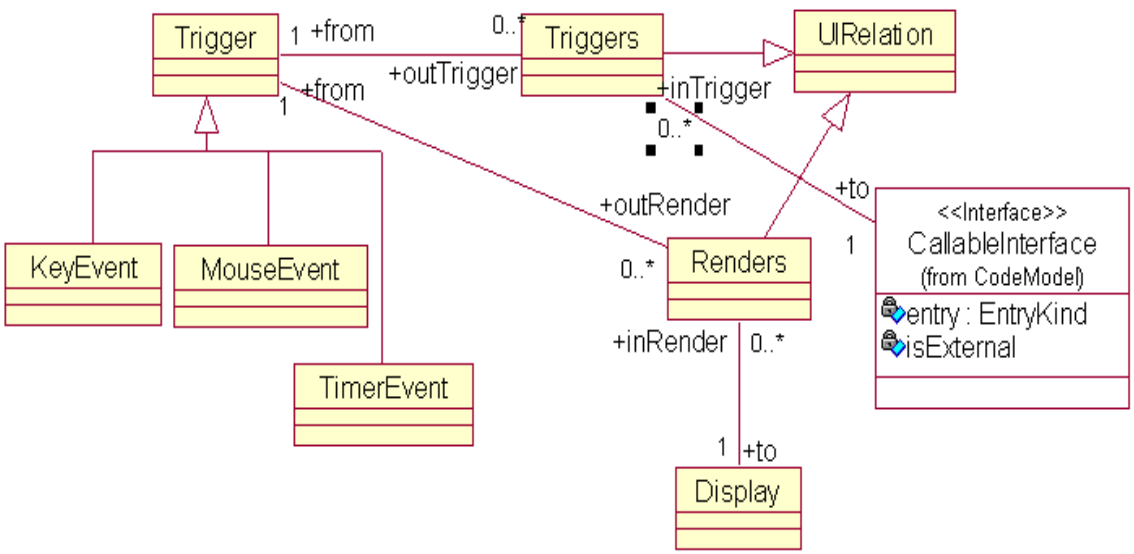
- This portion of the UI model describes lower-level behavior, linking events to the Display and to the application software (via CallableInterface)

UI events (cont.)



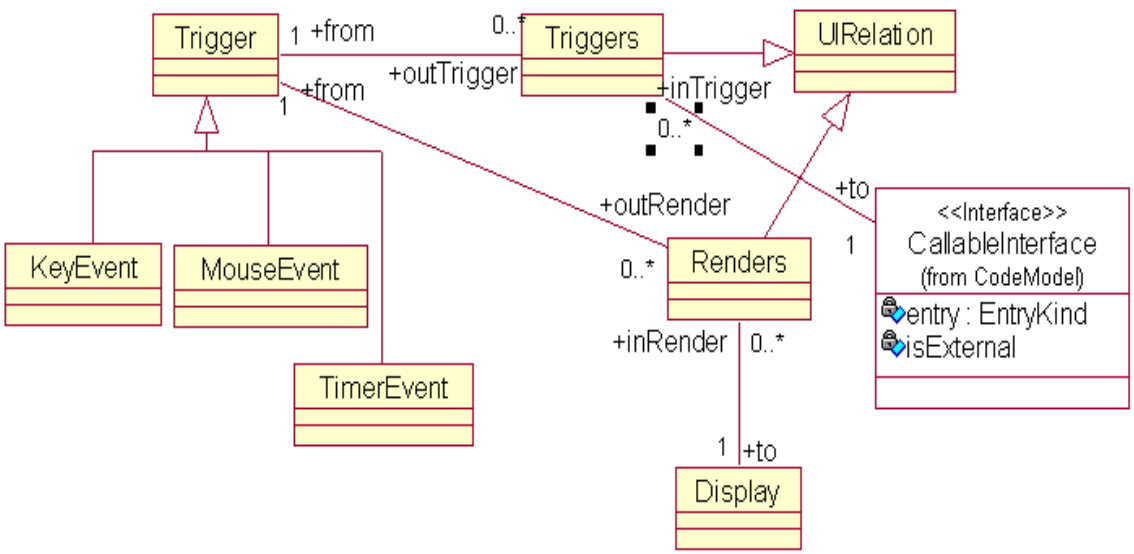
- A Trigger is an event in the UI which activates an application operation or changes the Display
- Three subtypes of Trigger
  - KeyEvent
  - MouseEvent
  - TimerEvent

UI events (cont.)



- When a Trigger leads to execution of application code, the Triggers as the relationship is used to connect to the CallableInterface from the CodeModel

UI events (cont.)



- When a Trigger does not lead to invocation of application code, the Trigger's Display changes are modeled using the Renders relationship class

User Interface Package

- Covers content and layout of UI
  - Relationships to Build package and Code package
- Covers behavior
  - Flow within UI model constructs
  - Relationships to Code package for software activation of UI, and for UI event-driven dispatching of application

Questions ?