Politechnika Krakowska
im. Tadeusza Kościuszki

Zakład Modelowania Systemów Złożonych

Paradygmat metamodelowania sterowanego kontekstem (CDMM-P)

Seminarium Katedry Inżynierii Oprogramowania (K7/W8)
Politechniki Wrocławskiej
i Sekcji Inżynierii Oprogramowania Komitetu Informatyki
Polskiej Akademii Nauk

dr inż. Piotr Zabawa
e-mail: pzabawa@pk.edu.pl
www: http://www.pk.edu.pl/~pzabawa

21.12.2016

Plan

- Wprowadzenie – dziedziny i pierwotny cel
- Współczesne problemy w stosowaniu Model-Driven Design
- Charakterystyka paradygmatu
  - Uogólnienie standardów MDA
  - Restryktywne ontologie otwarte w inżynierii oprogramowania
  - Definiowanie języka modelowania w czasie wykonania (run-time)
  - Możliwość wykorzystania dowolnego języka modelowania
  - Możliwość kastomizowania języków modelowania
  - Ułatwienie ewolucyjnego nadążania modeli za zmieniającym się językiem modelowania
  - Generowanie w czasie wykonania narzędzia do generowania kodu lub narzędzia do wykonywania modelu
- Wyzwania
- Podsumowanie

Wprowadzenie

- Słownictwo: grafowy język modelowania = metamodel; CDMM-F
- Dziedzina (w kontekście tej prezentacji)
  - Automatyzowanie procesów wytwórczych oprogramowania
  - Inżynieria oprogramowania (przeciwstawiana computer science)
  - Podejście Model-Driven Architecture
- Pierwotny cel – skonstruowanie uniwersalnego narzędzia do generowania kodu dla metodologii lat 80-tych
  - Główne kryterium projektowania rozwiązania – łatwość wprowadzania zmian (weak-coupling & strong-cohesion prowadzi do dekompozycji odpowiedzialności i zmiany ich podziału); ograniczenie zasięgu zmian
  - Pierwotna technologia: C++, PlugIn do Rational Rose oraz Rational Rose jako silnik dla własnej aplikacji, własne wzorce projektowe do IoC
  - Obecna technologia: Java, AspectJ, Spring, ByteBuddy – publikowanie od 2013 wyników prac rozpoczętych w 1993

Wprowadzenie

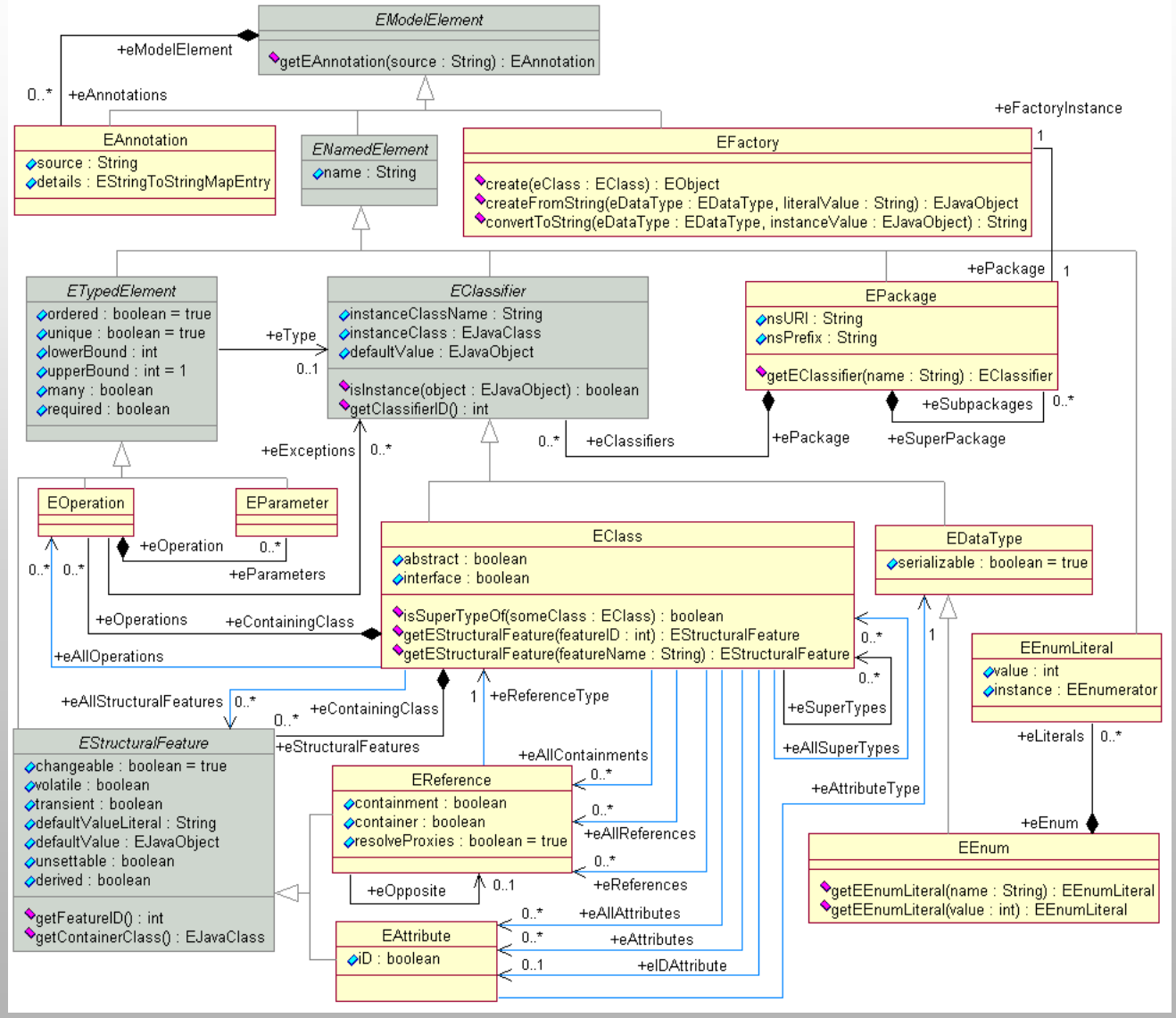
- Obserwacje
  - Konieczność usunięcia z klas meta-modelu relacji w formie referencji lub wskaźników
  - Przeniesienie wiązania ze sobą klas z compile-time do run-time
  - Utożsamienie metamodelu z warstwą danych (porównanie do UML ☺)
  - Możliwość definiowania klas encyjnych dla wierzchołków (encje wierzchołkowe) grafu i klas encyjnych dla krawędzi grafu (encje krawędziowe) – niepowiązane z niczym (100% brak zależności kompilacyjnych); możliwe warianty definiowania encji:
    - compile-time
    - run-time
    - mieszane
  - Tworzenie grafu w czasie wykonania i osadzanie w nim klas encyjnych
  - Wprowadzenie podziału odpowiedzialności w meta-modelu
  - Przeprowadzenie migracji odpowiedzialności

Wprowadzenie

- Zastosowania paradygmatu
  - Języki modelowania
  - Konstruowanie warstwy danych
  - Ontologie w modelowaniu wiedzy
  - Biblioteki grafowe
  - Parsing języków grafowych
- Perspektywy (przeanalizowana literatura)
  - Architektura oprogramowania
  - Ontologie
  - Paradygmaty
  - Domain-Specific Languages (DSL)
  - Mieszane
- *Czy zgubiłem jakąś dziedzinę?*

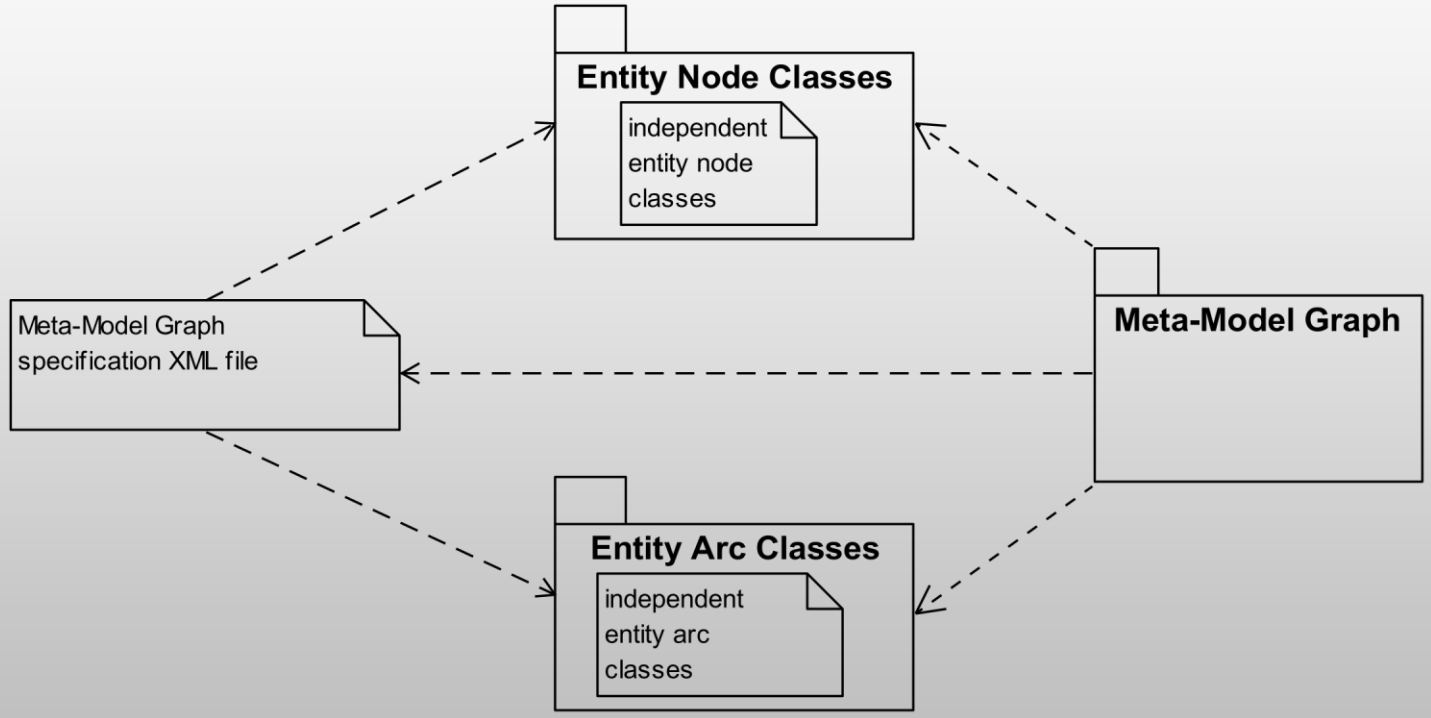
Problemy Model-Driven Design

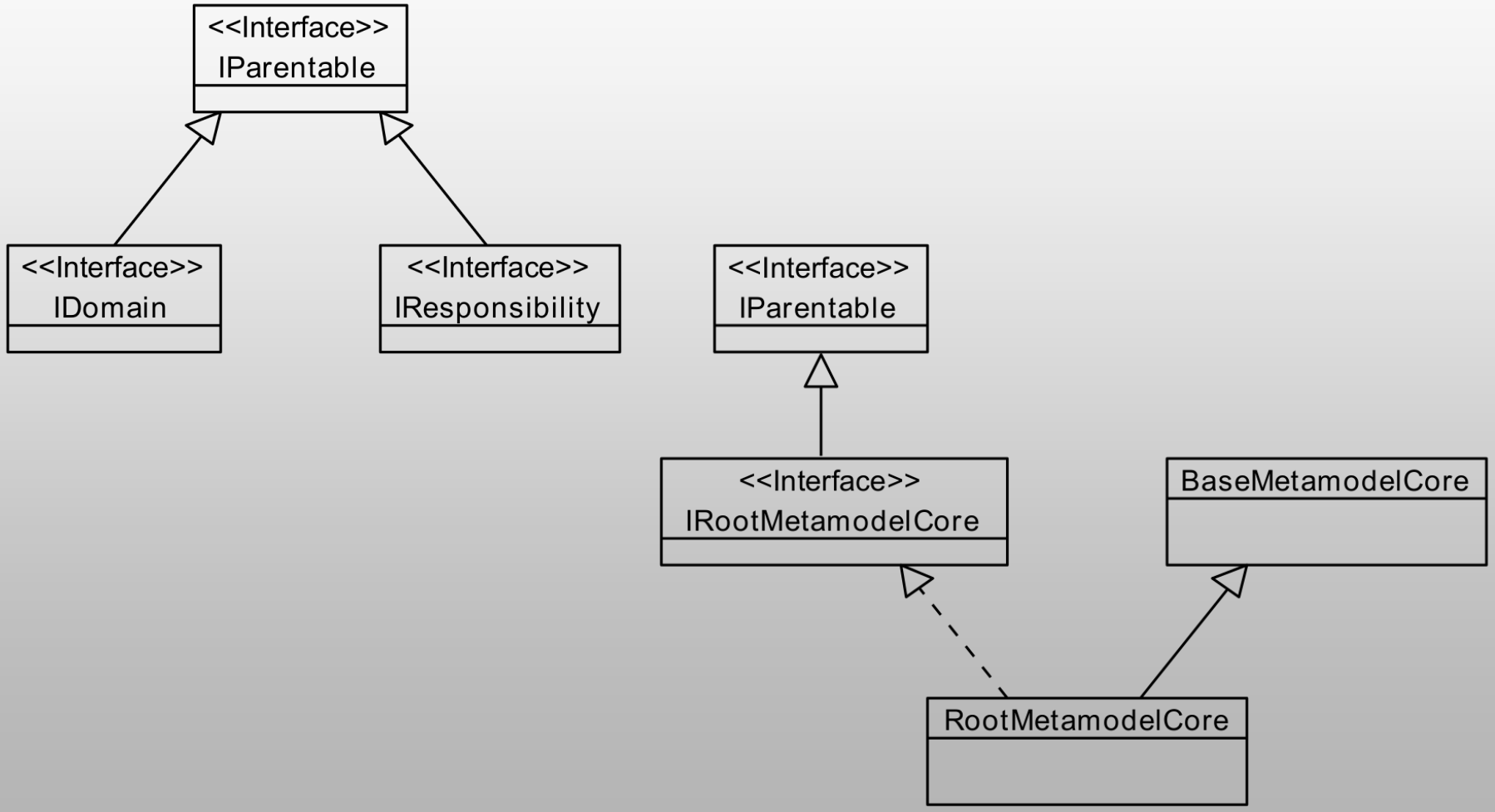
- Wieloletni i nie zawsze udany proces standaryzacji w OMG
  - Przykład 1: Aspect-Oriented Programming (próby od 2002 roku)
  - Przykład 2: relacja traits w językach na JVM – Scala, Groovy,…
- Cel standardu MDA – szybka odpowiedź na nowe technologie
- Wykorzystywanie paradygmatu klasowo-obiektowego w narzędziach
- Twórcy standardów MDA „zapomnieli” o prawidłowym ich zaprojektowaniu – brak dekompozycji odpowiedzialności jako dobrej praktyki z inżynierii oprogramowania
  - Przykład: eCore (wersja MOF dla Eclipse) – diagram!
- Brak możliwości uwzględnienia w standardach i narzędziach pojęć
  - Przykład: relacje N-arne (są tylko na diagramie, nie ma w metamodelu)
- Bardzo duża złożoność i rozmiar standardów OMG – oba parametry gwałtownie rosną; skutkuje to nie tylko trudnościami technicznymi ale niechęcią

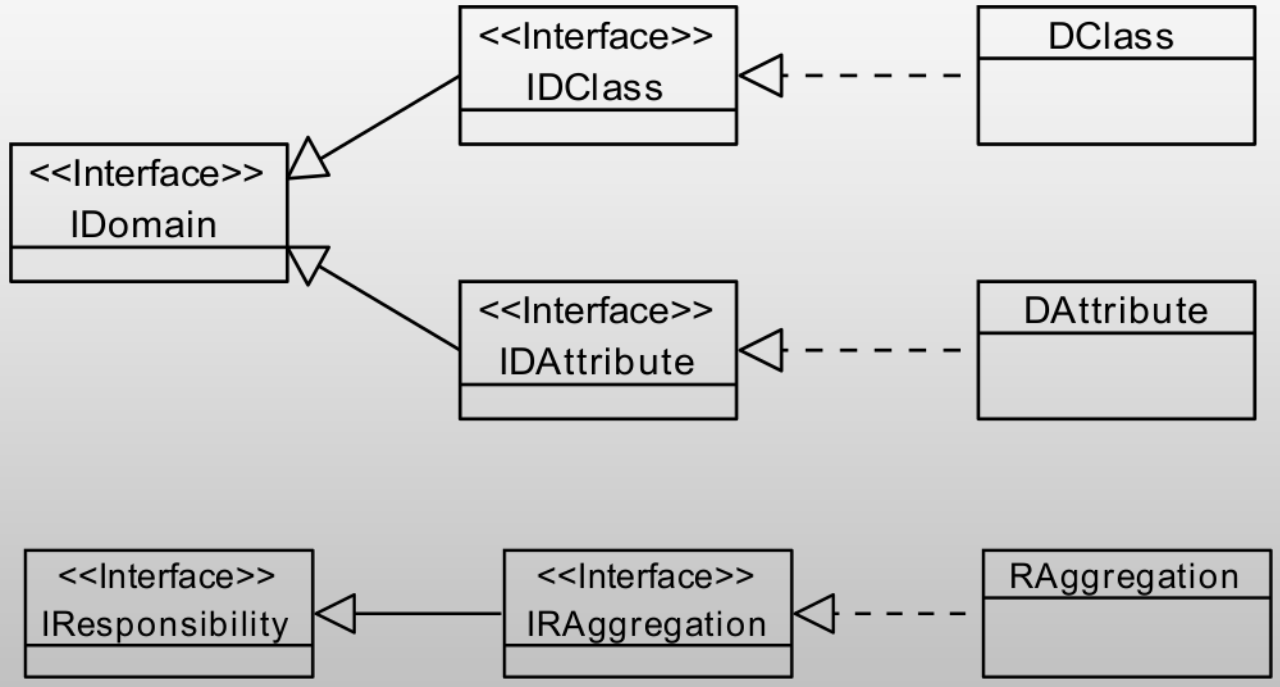

EModelElement
getEAnnotation(source : String) : EAnnotation
+eModelElement
0..*
+eAnnotations
EAnnotation
source : String
details : EStringToStringMapEntry
ENamedElement
name : String
EFactory
create(eClass : EClass) : EObject
createFromString(eDataType : EDataType, literalValue : String) : EJavaObject
convertToString(eDataType : EDataType, instanceValue : EJavaObject) : String
+eFactoryInstance
1
ETypedElement
ordered : boolean = true
unique : boolean = true
lowerBound : int
upperBound : int = 1
many : boolean
required : boolean
+eType
0..1
EClassifier
instanceClassName : String
instanceClass : EJavaClass
defaultValue : EJavaObject
isInstance(object : EJavaObject) : boolean
getClassifierID() : int
+ePackage
1
EPackage
nsURI : String
nsPrefix : String
getEClassifier(name : String) : EClassifier
+eSubpackages
0..*
+eSuperPackage
+eExceptions
0..*
0..*
+eClassifiers
+ePackage
EOperation
EParameter
+eOperation
0..*
+eParameters
0..*
0..*
+eOperations
+eContainingClass
+eAllOperations
EClass
abstract : boolean
interface : boolean
isSuperTypeOf(someClass : EClass) : boolean
getEStructuralFeature(featureID : int) : EStructuralFeature
getEStructuralFeature(featureName : String) : EStructuralFeature
EDataType
serializable : boolean = true
EEnumLiteral
value : int
instance : EEnumerator
+eLiterals
0..*
+eEnum
EEnum
getEEnumLiteral(name : String) : EEnumLiteral
getEEnumLiteral(value : int) : EEnumLiteral
0..*
0..*
+eSuperTypes
+eAllSuperTypes
1
+eAttributeType
+eAllStructuralFeatures
0..*
0..*
+eContainingClass
1
+eReferenceType
+eStructuralFeatures
EStructuralFeature
changeable : boolean = true
volatile : boolean
transient : boolean
defaultValueLiteral : String
defaultValue : EJavaObject
unsettable : boolean
derived : boolean
getFeatureID() : int
getContainerClass() : EJavaClass
+eAllContainments
0..*
EReference
containment : boolean
container : boolean
resolveProxies : boolean = true
0..*
+eAllReferences
0..*
+eReferences
+eOpposite
0..1
0..*
+eAllAttributes
0..*
+eAttributes
0..1
+eIDAttribute
EAttribute
iD : boolean

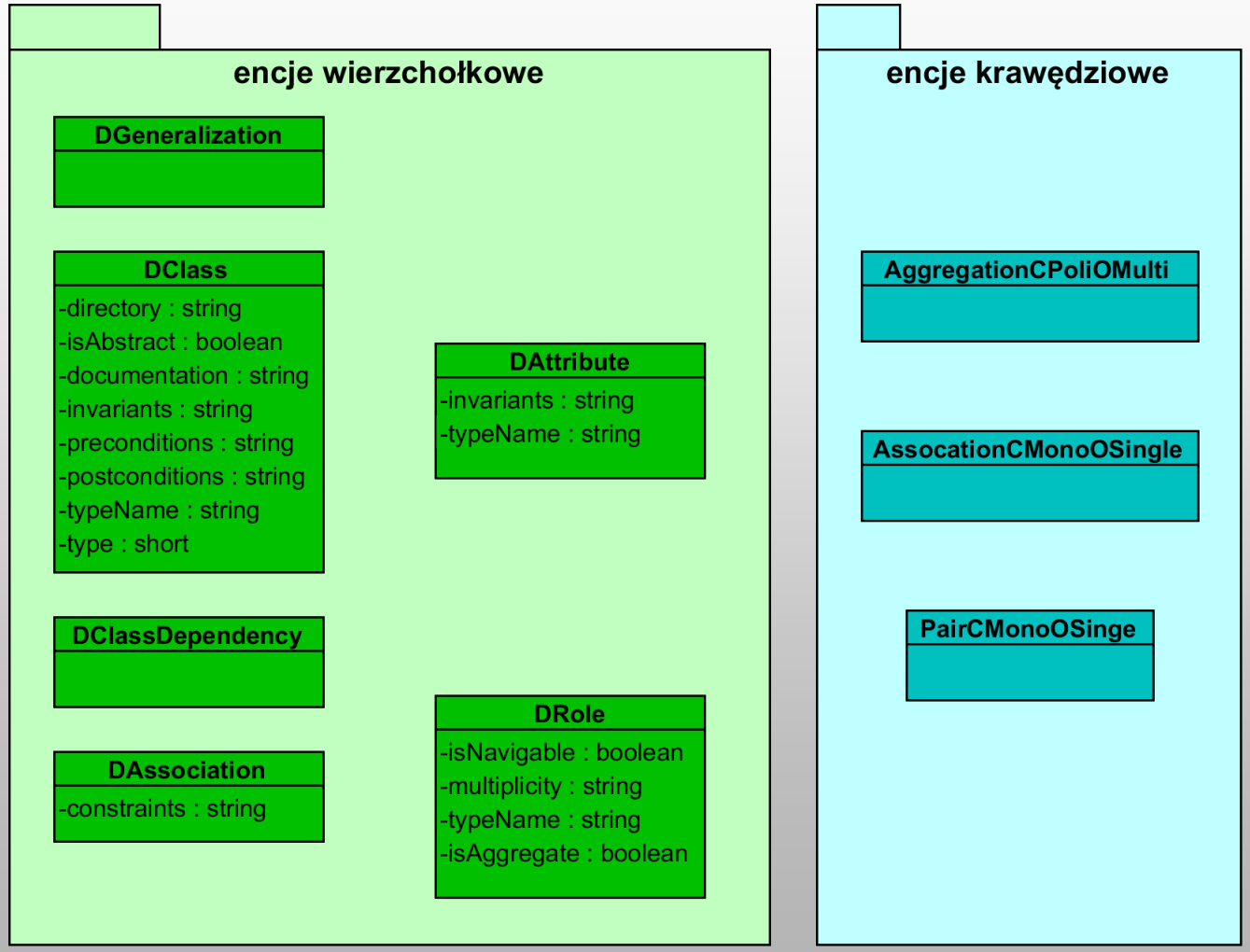
Charakterystyka paradygmatu

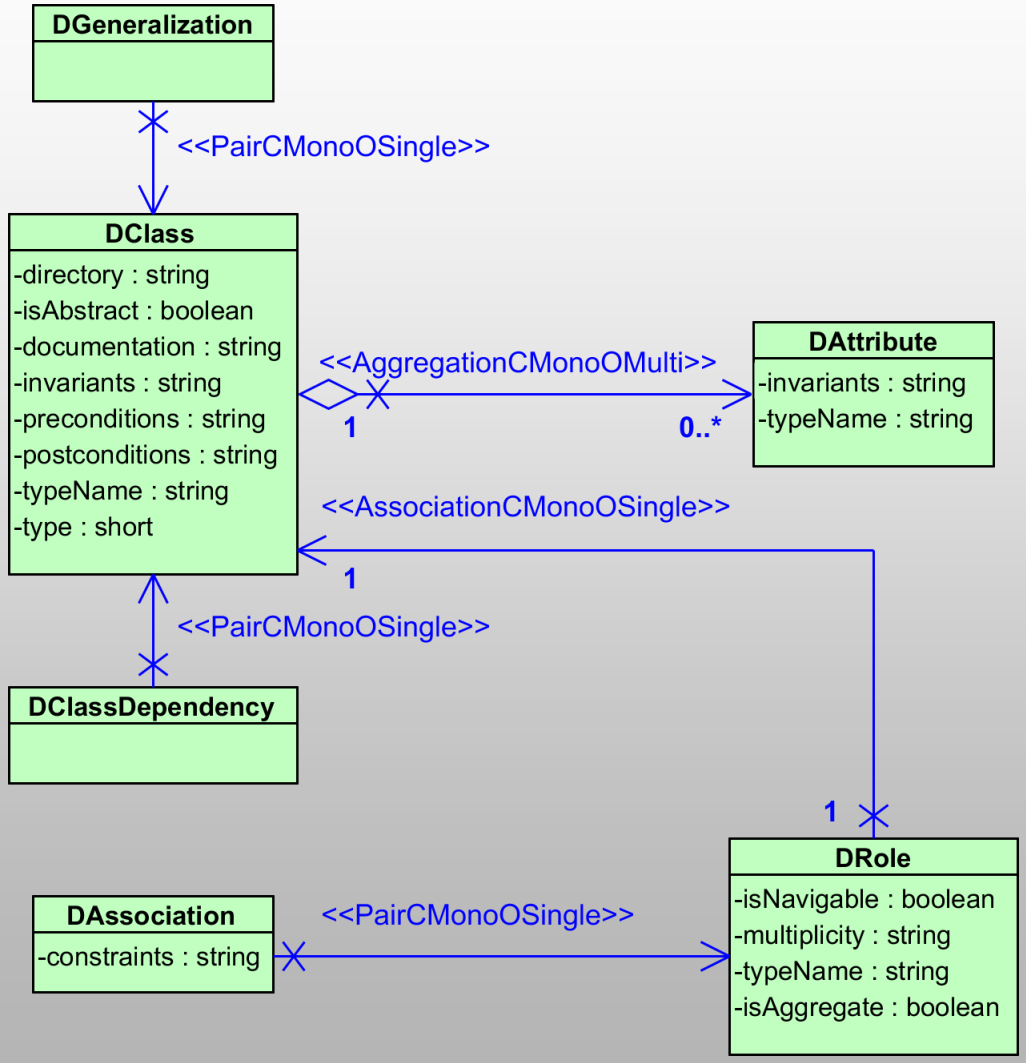
- Uogólnienie standardów MDA
  - Każdy standard MDA, w tym MOF jest instancją meta-modelu CDMM-P
- Restryktywne ontologie otwarte w inżynierii oprogramowania
  - Wbrew powszechnemu mniemaniu ontologie otwarte można stosować w inżynierii oprogramowania
  - CDMM-P na ontologie otwarte nakłada strukturę meta-modelu
- Definiowanie języka modelowania w czasie wykonania (run-time)
  - Ze względu na brak zależności kompilacyjnych strukturę grafu metamodelu można zdefiniować w czasie wykonania (plik XML z grafem) – diagram!
  - Klasy encyjne również można zdefiniować w czasie wykonania (pliki XML z encjami wierzchołków i encjami krawędzi)
- Możliwość wykorzystania dowolnego języka modelowania
  - Związek między istniejącymi (standardowymi?) językami modelowania a meta-modelami CDMM-P


Entity Node Classes
independent entity node classes
Meta-Model Graph specification XML file
Meta-Model Graph
Entity Arc Classes
independent entity arc classes


<<Interface>>
IParentable
<<Interface>>
IDomain
<<Interface>>
IResponsibility
<<Interface>>
IParentable
<<Interface>>
IRootMetamodelCore
BaseMetamodelCore
RootMetamodelCore


<<Interface>>
IDClass
DClass
<<Interface>>
IDomain
<<Interface>>
IDAttribute
DAttribute
<<Interface>>
IResponsibility
<<Interface>>
IRAggregation
RAggregation


encje wierzchołkowe
DGeneralization
DClass
-directory : string
-isAbstract : boolean
-documentation : string
-invariants : string
-preconditions : string
-postconditions : string
-typeName : string
-type : short
DAttribute
-invariants : string
-typeName : string
DClassDependency
DAssociation
-constraints : string
DRole
-isNavigable : boolean
-multiplicity : string
-typeName : string
-isAggregate : boolean
encje krawędziowe
AggregationCPoliOMulti
AssocationCMonoOSingle
PairCMonoOSinge


DGeneralization
<<PairCMonoOSingle>>
DClass
-directory : string
-isAbstract : boolean
-documentation : string
-invariants : string
-preconditions : string
-postconditions : string
-typeName : string
-type : short
<<AggregationCMonoOMulti>>
1
0..*
DAttribute
-invariants : string
-typeName : string
<<AssociationCMonoOSingle>>
1
<<PairCMonoOSingle>>
DClassDependency
1
DRole
-isNavigable : boolean
-multiplicity : string
-typeName : string
-isAggregate : boolean
DAssociation
-constraints : string
<<PairCMonoOSingle>>

Charakterystyka paradygmatu

- Możliwość kastomizowania języków modelowania
  - Można bez trudności wprowadzać zmiany strukturalne do standardowych języków modelowania (po ich konwersji do CDMM-P)
  - Można ograniczać standardowy język modelowania do potrzebnej części
  - Można rozbudowywać język modelowania o brakujące pojęcia
  - Można tworzyć własny język modelowania
- Generowanie w czasie wykonania narzędzia do generowania kodu i zasobów lub możliwość wykonywania modelu (jak w MDA – różne scenariusze)
- Ułatwienie ewolucyjnego nadążania modeli za zmieniającym się językiem modelowania (evolutionary meta-modeling)

Charakterystyka paradygmatu

- Szczegółowe cechy
  - Konstruowanie grafu metamodelu z pliku XML
  - Nawigowanie po metamodelu poprzez API frameworka CDMM-F
  - Tworzenie instancji metamodelu poprzez API frameworka CDMM-F
  - Graf skierowany
  - Tylko relacje asocjacyjne (UML: asocjacja, agregacja, kompozycja) w metamodelu (jak w modelu danych)
  - Operacje na klasach encyjnych tylko takie jak dla encji z warstwy danych
  - Kwestia generalizacji w metamodelu (dwa cele)
  - Nietypowe podejście do konstruowania API
  - Zastosowanie API opartego o paradygmat klasowo-obiektowy do CDMM-F wykraczającego poza ten paradygmat
  - Statyczność (znajomość metamodelu) vs. dynamiczność (brak znajomości metamodelu) API

Charakterystyka paradygmatu

- Szczegółowe cechy c.d.
  - Odmienne ryzyko testowania – szukanie wzorców ryzyk zależnych od implementacji CDMM-F (obecnie 64 możliwe wersje frameworka)
  - Odmienność testowania – specyfika zagadnienia i osłabione zależności; półautomatyczne generowanie testów scenariuszowych; parametryzowanie metamodelem == generowanie dziedziny

Charakterystyka paradygmatu

- Kwestie symetryzacji
  - Wstrzykiwanie encji krawędziowych do encji wierzchołkowych jako statycznych zobowiązań (symetryzacja) – diagram!
  - Możliwość wstrzykiwania hierarchicznych statycznych zobowiązań do encji wierzchołkowych i do encji krawędziowych (symetryzacja)
  - Możliwość ponownego użycia dowolnego fragmentu metamodelu (w tym dowolnego elementu metamodelu) w konstruowaniu innych języków modelowania (symetryzacja)
  - Asymetria: mało encji krawędziowych – przeciągnięcie odpowiedzialności (symetryzacja)
  - Nietypowe podejście do konstruowania API
  - Dostęp do elementów metamodelu za pomocą API poprzez każdy z rodzajów klas encyjnych (symetryzacja)

Charakterystyka paradygmatu

- Jak uzyskać dostęp do elementów metamodelu
  - Zastosowanie koncepcji akcesorów (*CDMM meta-model accessors*)

```
BaseClass getAccessor(node-entity-class, arc-entity-class)
```

gdzie:

`BaseClass` – klasa bazowa dla wszystkich elementów metamodelu

`node-entity-class` – klasa encji wierzchołkowej

`arc-entity-class` – klasa encji krawędziowej

Charakterystyka paradygmatu

- Jak uzyskać dostęp do elementów metamodelu c.d.
  - Metody dostępu do encji wierzchołkowej poprzez akcesory

```
void add(relationship-container, relationship-element)
T get(relationship-container)
int count(relationship-container)
List<T> getAll(relationship-container)
T get(relationship-container, index)
int count(relationship-container, relationship-element-class)
T get(relationship-container, relationship-element-class, index)
List<T> getAll(relationship-container, relationship-element-class)
```

gdzie:

relationship-container – obiekt encji wierzchołkowej poprzez który widoczne są jego obiekty encji krawędziowych łączących go w relacje z innymi obiektami encji wierzchołkowych;

relationship-element – obiekt encji wierzchołkowej skojarzony z obiektem encji krawędziowej łączącej go w relację z obiektem encji krawędziowej pozwalający uzyskać dostęp do ostatniego z tych obiektów poprzez relację uzyskaną z pierwszego z tych obiektów;

relationship-element-class – informacja o klasie obiektu encji krawędziowej;

index – indeks obiektu encji wierzchołkowej w kolekcji.

Test

```
ifc = Configuration.getCoreMetamodelAPIstatic(APP_CFG_FNAME);
// entity node DClass object creation
DClass cls = (DClass) ifc.getDomain(DClass.class, "class");
// entity node DAttribute object creation
DAttribute attr = (DAttribute) ifc.getDomain(DAttribute.class,
  "attribute");
// getting access to the DClass class accessor
IRAggregation classAccessor = (IRAggregation) ifc.getAccessor(
  DClass.class, IRAggregation.class);
// adding entity node object to its accessor
classAccessor.add_IRAggregation(cls, attr);
// getting access to the entity node object indexed as i in relation
// through its accessor
DAttribute att = classAccessor.get_IRAggregation(cls, i);
```

Wyzwania

- Żadna technologia nie wspiera intensywnego wykorzystywania statyki
- Ograniczenia technologii związanych z JVM
  - Maszyny wirtualnej JVM
  - Języka programowania Java (i innych na JVM)
  - AspectJ
  - Springa
- Ograniczenia standardów
  - Braki w standaryzacji Java EE w zakresie kontekstu aplikacji – diagram!
- Brak wsparcia ORM i OGM
- Opłacalność opracowania własnych technologii
- Liczne warianty implementacji paradygmatu (dziesiątki w ramach technologii Java), kwestia uniwersalności API
- Brak powiązania meta-programowania z meta-modelowaniem

Seam Framework - JBoss Seam - Mozilla Firefox

Seam Framework - JBoss Seam

seamframework.org

Design

Built with Seam

You can find the full source code for this website in the Seam package in the directory /examples/wiki. It is licensed under the LGPL.

Created with and running on: Seam

Seam Drools	Drools Project
Seam Faces	Apache DeltaSpike and JSF 2.2
Seam Errai	Errai
Seam International	Apache DeltaSpike
Seam JCR	ModeShape
Seam JMS	JSR 343
Seam Mail	cdi-mail
Seam Persistence	Apache DeltaSpike and JTA 1.1
Seam Remoting	Deprecated
Seam Reports	TBD
Seam Rest	RESTEasy
Seam Security	PicketLink
Seam Servlet	CDI 1.1
Seam Social	Agorava
Seam Spring	Proposed for Apache DeltaSpike
Seam Validation	Bean Validation 1.1
Seam Wicket	Apache Wicket
PDF and Spreadsheets	TBD

The Seam Framework - Next generation enterprise Java development

Seam is a powerful open source development platform for building rich Internet applications in Java. Seam integrates technologies such as Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), Java Persistence (JPA), Enterprise Java Beans (EJB 3.0) and Business Process Management (BPM) into a unified full-stack solution, complete with sophisticated tooling.

Seam has been designed from the ground up to eliminate complexity at both architecture and API levels. It enables developers to assemble complex web applications using simple annotated Java classes, a rich set of UI components, and very little XML. Seam's unique support for conversations and declarative state management can introduce a more sophisticated user experience while at the same time eliminating common bugs found in traditional web applications.

Learn more

To find out more about Seam, read this introduction and check out some tutorial code examples. If you're evaluating Seam, please see our answers to frequently asked questions.

If you are a new Seam user, follow this roadmap to get started quickly. If you want to contribute to Seam, register on this website and join the Seam Community. If you are interested in development of Seam, see the list of open issues and tasks by priority.

Seam is licensed under the terms of the LGPL. Full commercial support is available. Eclipse-based tooling for Seam applications is provided by JBoss Tools.

Reinvesting in Java EE

Seam is based on the Java EE platform. That's why reinvestment in Java EE standards is crucial to Seam's future. Many ideas which originated in the Seam ecosystem are being contributed back to the Java EE specification by Red Hat, being refined in the process. The most notable initiative is JSR-299: Java Context and Dependency Injection. This website is home of the reference implementation, Weld. Check out the development page to get involved. The Seam community also has lots of ideas for JSF 2.0 and ~~JSF 2.1~~, so check out those pages to keep up with the proposals.

Podsumowanie

- Prezentacja opracowanego przeze mnie nowego, częściowo zweryfikowanego podejścia do konstruowania języków modelowania i narzędzi
- Na seminarium oczekuję sprzężenia zwrotnego – komentarzy, pytań, sugestii
- Jestem otwarty na współpracę w zakresie
  - opracowywania zagadnień teoretycznych
  - rozwijania narzędzi
  - wdrożeń

Koniec