An Ontological Approach to Domain Engineering

Richard de Almeida Falbo, Giancarlo Guizzardi, Katia Cristina Duarte
International Conference on Software Engineering and Knowledge Engineering, SEKE’02

Taehoon Song
2007. 7. 11
Contents

- Introduction
- Background
- Ontology-based Domain Engineering (ODE)
- Applying ODE in software quality domain
- Related work
- Conclusions
- Discussion
Introduction (1/3)

† To get the main benefits of software reuse, What we need?
   † Develop *for* reuse
   † Develop *with* reuse

† Domain engineering
   † Concerns developing *for* reuse

† Ontologies
   † Can play an important role in this context
   † An *ontology*
     † Can promote common understanding among developers
     † Can be used as a domain model
What is an ontology?
- Set of concepts with a domain
- Relationships between those concepts
- ex. Card

One of the major drawbacks to a wider use of ontologies in software engineering:
- Lack of approaches to insert ontologies into a more conventional software development process
Introduction (3/3)

- To put ontologies in practice,
  - Need an approach to derive **object models** from **ontologies** in order to derive widely reusable assets
    - Because current leading paradigm in software engineering is the object-oriented technology

- Research goal
  - Propose an ontology-based approach to domain engineering
    - **Building ontologies**
    - **Deriving object frameworks from them**
Domain engineering

Concerns the work

- Required to establish a set of software artifacts

Purpose

- Identify, model, construct, catalog and disseminate a set of software artifacts
  - Can be applied to existing and future software in a particular application domain

Main activities

1. Domain analysis
2. Infrastructure specification
3. Infrastructure implementation
Domain analysis

Involves identification, acquisition and analysis of domain knowledge

Purpose

- Produce a domain model
  - Serves an unified source of reference
  - Serves a repository of the shared knowledge
  - Serves a specification to the developer of reusable components

However,

- There exists a gap between
  - The kinds and forms of the domain knowledge in a domain model
  - The content and form of software assets

Therefore, need to build a reuse infrastructure
Background (3/4)

- **Infrastructure specification**
  - Involves
    - Selecting and organizing the reusable information
    - Determining
      - how it should be packaged into components
      - how these components should be indexed
  - **Purpose**
    - Define the aspects of the problem domain
      - Should be supported by the component repository in order to achieve the reuse system requirements

- **Infrastructure implementation**
  - Use infrastructure specification with the semantics captured by the domain model
  - Actually produces and tests the components
Ontologies in domain engineering

Applications of ontologies can be classified

1. Neutral authoring
   - An ontology for defining different approaches

2. Ontology as specification
   - An ontology of a given domain
     - Provides a vocabulary for specifying requirements for target applications
       - A basis for specification and development of domain applications, allowing knowledge reuse

3. Common access to information
   - To avoid misunderstanding among developers

4. Ontology-based search
   - To reduce the overall amount of time spent searching
Ontology-based domain engineering (1/4)

A systematic approach for building ontologies

- Purpose Identification and Requirement Specification
- Ontology Capture
- Ontology Formalization
- Evaluation and Documentation
- Integrating Existing Ontologies

Figure 1. Steps in the ontology development process

Competency questions

Graphical language

Design quality criteria:
- Clarity
- Coherence
- Extendibility
- Minimal encoding bias
- Minimal ontological commitment

: Constant interaction, albeit weaker, between the associated steps

: Main work flow

General modular ontologies

Figure 1.

Steps in the ontology development process
Ontology-based domain engineering (2/4)

- A systematic approach for building ontologies (cont’d)
  - LINGO; Graphical Language for Expressing Ontologies (GLEO)
    - Meta-ontology
      - Defines the basic notations to represent a domain conceptualization
  - Axioms
    - Mereological theory
      - Means ‘science or theory of parts’: μεροσ
        - Mereological relation: part-of decomposition
        - Mereological ontology: defines the part-of relation and its properties
    - Axioms in on ontology
      - Derivation axioms: New information from previously existing knowledge
        - Epistemological axioms: Structuring of the concepts and relations
        - Ontological axioms: Domain signification constraints
      - Consolidation axioms: Integrity law
        - Define constraints for establishing a relation
        - Define an object as an instance of a concept
Ontology-based domain engineering (3/4)

- From domain ontologies to objects
  - Systematic approach
    1. A set of directives
      - To guide the mapping
    2. Design patterns
    3. Transformation rules

Mapping of ontological and consolidation axioms

In set theory,

\[ \text{contract} = ((\text{Organization}, \text{Person}, \text{contract}(x,y))) \]

The epistemological structures of the domain ontology
(concepts, relations, properties and roles)

Their counterparts in the object-oriented paradigm
(classes, associations, attributes and roles)

12/20
Ontology-based domain engineering (4/4)

- From domain ontologies to objects (cont’d)
  - Set-based formalism
    - In set theory, some essential operations
      - ∩, ~, ∈, ∀, Im …
    - Mapping of relations
  - Set of transformation rules

<table>
<thead>
<tr>
<th>T0</th>
<th>∀ x:X, ∀ y:Y r1(x,y) ↔ y ∈ C ⇒ Im+(x, r1):Type = C, such that if # Im(x, r1) = 1 then Type = Y else Type = Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Im+(x, r1) ⇒ x.r1()</td>
</tr>
<tr>
<td>T2</td>
<td>Im+(y, r1~) ⇒ y.r1()</td>
</tr>
<tr>
<td>T3</td>
<td>r1(x,y) ⇒ x.r1()</td>
</tr>
<tr>
<td>T4</td>
<td>r1(x) ⇒ x.r1()</td>
</tr>
<tr>
<td>T5</td>
<td>A SetTheoryOperation a ⇒ A.SetTheoryOperationImplementation(a)</td>
</tr>
<tr>
<td>T6</td>
<td>Im(A, r1) ⇒ Set.Im(A,&quot; r1&quot;)</td>
</tr>
</tbody>
</table>
| T7                        | x.r1():Y≡C ⇒ public class X{
                      | public Y r1(){
                      | return C;
                      |
                      |}                                                                                                                             |
Applying *ODE* in software quality domain (1/4)

- **A software quality ontology**
  - We must understand **what software quality means**
  - But this is **not** an easy task

- **Competency questions:**
  1. Which is the nature of a quality characteristic?
  2. In which sub-characteristics can a quality characteristic be decomposed?
  3. Which characteristics are relevant to evaluate a given characteristic?
  4. To which paradigm a quality characteristic is applicable?

- **Classifications**
  1. If a quality characteristic can be directly measured or not
  2. Enforce that product characteristics should only be used to evaluate software artifacts
Applying *ODE* in software quality domain (2/4)

- A software quality ontology (cont’d)
  - Part of the software quality ontology

![Diagram of software quality ontology]

- Several consolidation axioms
  - Subsumption theory: Axioms (E1~E3)
  - Whole-part relation: Axioms (E4~E7)
  - Cardinality constraints: Axioms (E8~E10)
Applying *ODE* in software quality domain (3/4)

- **Object-based domain reuse infrastructure**
- **Part of the knowledge package**

- All classes derived directly from the ontology:
- Remainder class are from the **Whole-Part design pattern** used.

✓ Several consolidation axioms

\[
(\forall qc,qc_1)(\text{subqc}(qc_1,qc) \land \text{prodqc}(qc) \rightarrow \text{prodqc}(qc_1))
\]

✓ Several ontological axioms

\[
(\forall qc \rightarrow (\exists p)(\text{applicability}(qc,p) \rightarrow \text{pdgInd}(qc))
\]

- Translated to the **set-theory formalism**
- Mapped to method **pdgInd()**
Applying *ODE* in software quality domain (4/4)

- Object-based domain reuse infrastructure (cont’d)

1. Several consolidation axioms
   \((\forall qc,qc_1)(\text{subqc}(qc_1,qc) \land \text{prodqc}(qc) \Rightarrow \text{prodqc}(qc_1))\)

   - Implemented by the method `addSuperQC`

   ```java
   addSuperQC(KNonMeasurableQC: qc): boolean
   {
       boolean result = false;
       if (result = (qc.isProductQC && this.isProductQC))
       {
           superQC.add(qc);
           qc.addSubQC(this);
       }
       return result;
   }
   ```

2. Several ontological axioms
   \((\forall qc) \neg(\exists p)(\text{applicability}(qc,p) \Rightarrow \text{pdgInd}(qc))\)

   - Transformation rules:

   ```java
   1. \forall qc: \text{QualityCharacteristic} \ pdgInd(qc, True) \leftrightarrow 
      \#\text{Im}_+(qc, \text{applicability}) = 0
   2. \text{qc.pdgInd()}: \text{Boolean} \equiv \#\text{Im}_+(qc, \text{applicability}) = 0
   3. \text{qc.pdgInd()}: \text{Boolean} \equiv (qc.\text{applicability()}).\text{card()} = 0
   4. public class KQualityCharacteristic
      {
          public Boolean pdgInd()
          {
              return (this.\text{applicability()}).\text{card()} == 0;
          }
      }
   ```

   - \(2.\text{T0}\)  \(3.\text{T1, T5}\)  \(4.\text{T7}\)
Related work (1/2)

- **Feature Oriented Domain Analysis (FODA, 1990)**
  - Way of capturing commonality and variability information. Features are captured in a feature model.

  - Systematic model-driven approach
    - To domain-specific, object-oriented software reuse
      - Use case models are central to all steps in RSEB
  - FeatuRSEB (1998)
    - FODA’s feature model was adapted to RSEB
      - To solve RSEB’s problem

- **Organization Domain Modeling (ODM, 1998)**
  - Multi-modeling approach based on a mathematical formalism (Sigma)
    - To define the core domain modeling process in a manner independent of assumptions about the specific modeling representation used
## Related work (2/2)

### Comparison

<table>
<thead>
<tr>
<th>Aspects</th>
<th>FODA</th>
<th>RSEB</th>
<th>ODM</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent of reuse technology</td>
<td>-</td>
<td>-</td>
<td>Ontology-based approach to domain analysis</td>
<td></td>
</tr>
<tr>
<td>Capturing domain functionality</td>
<td>Use-case centered</td>
<td>-</td>
<td>-</td>
<td>Task ontology</td>
</tr>
<tr>
<td>Define formal axioms</td>
<td>FeatuRSEB: resemble a data dictionary</td>
<td>-</td>
<td>Formal axioms (for automated tools)</td>
<td></td>
</tr>
<tr>
<td>Capturing domain conceptualization</td>
<td>-</td>
<td>-</td>
<td>Condition</td>
<td>LINGO, Axioms and properties</td>
</tr>
</tbody>
</table>
Conclusions and discussion

- This paper shows
  - Ontology-based Domain Engineering (ODE)
    - LINGO
    - Axioms
  - Ontological approach for domain engineering
    - Mapping ontological axioms to the object model

- Discussion
  - Need to clarify the steps of ODE for automated tools
  - Using UML for domain engineering
    - Need to define a subset of UML
      - Using OCL or ignoring the OCL constraints
    - Need axiomatic approaches for using UML
Abstract

- **Domain engineering**
  - Aims to support systematic reuse
  - Focus on modeling common knowledge in a problem domain

- **Ontologies**
  - Hold great promise for software reuse

- **Paper contents**
  - Ontology-based Domain Engineering (*ODE*)
  - Ontological approach for domain engineering
    - Aims to join ontologies and object-oriented technology
In philosophy, Ontology

From the Greek ὄν, genitive ὄντος: of being (part. of ἐἶναι: to be) and -λογία: science, study, theory

Is the study of being or existence and forms the basic subject matter of metaphysics

- Metaphysics
  - Branch of philosophy concerned with explaining the ultimate nature of reality, being, and the world

It seeks to describe or posit

- Basic categories and relationships of being or
- Existence to define entities and types of entities within its framework
Software process

Common process framework

Framework activities

Task sets

Tasks

Milestones, deliverables

SQA points

Umbrella activities

Define a small number of framework activities
LINGO (1/2)

- **Notation**
  - Ex. part of the activity ontology

- **Axioms**
  - Concept of pre-activity
    \[(\forall a_1, a_2)(\text{preactivity}(a_1, a_2) \iff (\exists s)(\text{input}(s, a_2) \land \text{output}(s, a_1)))\]
  - Activity(a,t)
    \[(\forall a)(\text{activity}(a, \text{CertificationAct}) \lor \text{activity}(a, \text{TestAct}) \Rightarrow \text{activity}(a, \text{QualityAct}))\]
  - Consolidation axioms
    \[(\forall a_1, a_2)(\text{preactivity}(a_1, a_2) \Rightarrow \text{activity}(a_1, *) \land \text{activity}(a_2, *))\]

*: value of argument
Ontology design quality

Minimum ontology commitment

Must embody only notations that are necessary to express ontologies

Ontology commitment

For a set of agents
- They can communicate about a domain of discourse without necessarily operating on a globally shared theory
- Based on the Knowledge-Level perspective
  - Description of the knowledge of an agent
    - Independent of the symbol-level representation used internally by the agent
  - *Tell and ask* functional interface
    - Client interacts with an agent by making logical assertions (tell) and posing queries (ask)

Formal model of domain

Describe a formal semantics
Hybrid formalism (1/4)

- Theoretical foundation for a Set-based language
  - Set theory
    - Collections of zero or more elements whose members are unique and their order is immaterial
    - Ex. Statement $x \in \text{Mortal}$ commits $x$ to the concept $\text{Mortal}$, both intentionally and extensionally
      - N-ary relation: $R = (C_1, C_2...C_n, p(x_1, x_2...x_n))$
        - $C_i$: a different set involved in the relation
        - $p(x_i)$: functional predicate open $n$ variables
          - that maps each element from the cross-product $C_1 \times C_2 \times ... C_n$ onto a boolean value
      - $R^*$ (solution set): subset of $C_1 \times C_2 \times ... C_n$ whose all members $e_i$ satisfy the predicate $p(e_i)$
  - Essential operations
    - Relations between sets, Properties of sets, Restriction on relations
    - Relations between sets and their members, Basic logical operators, Quantifiers
    - set relational Image ($Im$), element relational image ($Im^+$)
Hybrid formalism (2/4)

- **Set framework**

  Support framework that plays a fundamental role in ontology-to-Java objects mapping process

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation prototype</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \subseteq )</td>
<td>A.contains (B)</td>
<td>Verify if set B is contained in set A</td>
</tr>
<tr>
<td>( = )</td>
<td>A.equals (B)</td>
<td>Verify if set B is equals to set A</td>
</tr>
<tr>
<td>( \cup )</td>
<td>A.union (B)</td>
<td>Returns the set ( A \cup B )</td>
</tr>
<tr>
<td>( \cap )</td>
<td>A.intersection (B)</td>
<td>Returns the set ( A \cap B )</td>
</tr>
<tr>
<td>( # )</td>
<td>A.cardinality ()</td>
<td>Number of elements of set A</td>
</tr>
<tr>
<td>{C \mid C \subseteq A}</td>
<td>A.subset(&quot;C&quot;)</td>
<td>Returns the set C if C is a subset of A</td>
</tr>
<tr>
<td>( \setminus )</td>
<td>A.difference(B)</td>
<td>Returns the difference between two sets</td>
</tr>
<tr>
<td>( \in )</td>
<td>A.in(x)</td>
<td>Verify if the element x belongs to set A</td>
</tr>
<tr>
<td>( + )</td>
<td>A.add(x)</td>
<td>Adds the element x to the set A</td>
</tr>
<tr>
<td>( - )</td>
<td>A.remove(x)</td>
<td>Removes the element x from the set A</td>
</tr>
<tr>
<td>( \text{Im} )</td>
<td>Set.Im(a,r1)</td>
<td>Returns the set ( \text{Im}(a,r1) )</td>
</tr>
<tr>
<td>( \text{Im} )</td>
<td>Set.Im(A,r1)</td>
<td>Returns the set ( \text{Im}(A,r1) ) where ( A = {a_1,...,a_n} )</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Set.select(&quot;w&quot;,op,&quot;z&quot;)</td>
<td>Returns the selection ( \sigma_{w _op _z}(A) )</td>
</tr>
</tbody>
</table>
Hybrid formalism (3/4)

Framework that implements the mathematical type Set

Set

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set()</td>
</tr>
<tr>
<td>Set(SetElement[] e)</td>
</tr>
<tr>
<td>in(SetElement x):boolean</td>
</tr>
<tr>
<td>contains(Set s):boolean</td>
</tr>
<tr>
<td>equals(Set s):boolean</td>
</tr>
<tr>
<td>union(Set s):Set</td>
</tr>
<tr>
<td>intersection(Set s):Set</td>
</tr>
<tr>
<td>cardinality():int</td>
</tr>
<tr>
<td>difference(Set s):Set</td>
</tr>
<tr>
<td>extension():Iterator</td>
</tr>
<tr>
<td>SubSet(String subset):Set</td>
</tr>
<tr>
<td>get_Instance(Object key):SetElement</td>
</tr>
<tr>
<td>add(SetElement x)</td>
</tr>
<tr>
<td>remove(SetElement s)</td>
</tr>
<tr>
<td>select(String prop,String operator, Object value):Set</td>
</tr>
<tr>
<td>Set.Element(Object o,String relation):Set</td>
</tr>
<tr>
<td>Set.Element(Set s,String relation):Set</td>
</tr>
</tbody>
</table>

<<SetElement>>

get_Key():Object
equals(SetElement s):boolean

MemberSet

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberSet(String id)</td>
</tr>
<tr>
<td>MemberSet(String id,SetElement[] e)</td>
</tr>
</tbody>
</table>

PersistentSet

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersistentSet(String a_source)</td>
</tr>
<tr>
<td>PersistentSet(String a_source,SetElement[] e)</td>
</tr>
<tr>
<td>store():void</td>
</tr>
<tr>
<td>retrieve():void</td>
</tr>
</tbody>
</table>
Hybrid formalism (4/4)

- Ex. Part & axioms of a Software Process Ontology (SPO)

<table>
<thead>
<tr>
<th>Id</th>
<th>Axiom</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA1</td>
<td>( \forall a ) constructionActivity(a) \rightarrow activity(a)</td>
<td>Epistemological</td>
</tr>
<tr>
<td>EA2</td>
<td>( \forall a ) managementActivity(a) \rightarrow activity(a)</td>
<td>Epistemological</td>
</tr>
<tr>
<td>EA3</td>
<td>( \forall a ) qualityAssuranceActivity(a) \rightarrow activity(a)</td>
<td>Epistemological</td>
</tr>
<tr>
<td>EA4</td>
<td>( \forall a_1, a_2, a_3 ) subActivity(a_1, a_2) \land \text{subActivity}(a_2, a_3) \rightarrow \text{subActivity}(a_1, a_3)</td>
<td>Epistemological</td>
</tr>
<tr>
<td>EA5</td>
<td>( \forall a_1, a_2 ) subActivity(a_1, a_2) \rightarrow \neg \text{subActivity}(a_2, a_1)</td>
<td>Epistemological</td>
</tr>
<tr>
<td>CA1</td>
<td>( \forall a, s ) input(s, a) \rightarrow artifact(s) \land activity(a)</td>
<td>Consolidation</td>
</tr>
<tr>
<td>CA2</td>
<td>( \forall a, r ) usage(r, a) \rightarrow resource(r) \land activity(a)</td>
<td>Consolidation</td>
</tr>
<tr>
<td>OA1</td>
<td>( \forall a ) composedActivity(a) \leftrightarrow \exists a_1 \text{subActivity}(a_1, a)</td>
<td>Ontological</td>
</tr>
<tr>
<td>OA2</td>
<td>( \forall a, a_1, r ) (subActivity(a_1, a) \land usage(r, a_1)) \rightarrow usage(r, a)</td>
<td>Ontological</td>
</tr>
</tbody>
</table>
Exceptions for ontology to objects mapping

- Some concept
  - Can be better mapped to attribute of a class
    - Because they do not have a meaningful state in the sense of an object model
- Some concept
  - Should not be mapped to an object model
    - Because they were defined only to clarify some aspect of the ontology,
    - But they do not enact a relevant role in an object model
- Relations involving a concept
  - That is mapped to an attribute (or that is not considered in the mapping)
    - Should not be mapped to the object model
Whole-part design pattern

<<SetElement>>

<<IWhole>>
whole():Whole

<<IPart>>
part():Part

A
b():B
setB(IPart b):boolean
removeB(IPart b)

Whole
parts:Set
whole():IWhole
asymmetry(IPart c):boolean
specConst(IPart c):boolean
part0f(Itodo w,IParte c)
part():Set
setPart(IPart c)
removePart(IPart c)

Aggregation
disjoint:Set
specConst(IPart c):boolean
disjointness(IPart c):boolean
setDisjoint(IWhole w)

Composition
specConst(IPart c):boolean
exclusiveness(IPart c):boolean

Part
whole:Set
whole():set
setWhole(IWhole w)
removeWhole(IWhole w)

public boolean setPart(IPart c)
{
    boolean result = false;
    if (acyclicity(c) && SpecConstraint(c))
    {
        result=true;
        part.add(c);
        (c.part()).setWhole(whole);
    }
    return result;
}
## Feature Oriented Domain Analysis (FODA)

### Cons
- **Semantics of feature is not precisely defined**
  - Not formally clear what is meant by “feature”

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select Features from Domain</td>
<td>Identify desired features from features model, i.e., operations, context, and representation.</td>
<td>List of desired features</td>
</tr>
<tr>
<td>2. Create Object Specifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Identify Objects</td>
<td>Identify data items maintaining state or requiring explicit control.</td>
<td>Initialized Object Form, Entity List</td>
</tr>
<tr>
<td>2. Derive Object Operations and</td>
<td>Analyze features model for operation variations based on alternatives or context and shown in operational model.</td>
<td>Completed Object Form</td>
</tr>
<tr>
<td>Inputs/Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Create Subsystem Specifications</td>
<td>Group together objects that work together, correlated to set of related features in features model.</td>
<td>Completed Subsystem Form</td>
</tr>
<tr>
<td>4. Create Surrogate Specifications for Devices</td>
<td>Determine external interfaces for applications and determine their control and data characteristics.</td>
<td>Initialized Surrogate Form</td>
</tr>
</tbody>
</table>
Reuse-driven Software Engineering Business (RSEB)

- **Cons**
  - Incomplete with respect to domain analysis
  - Not provide explicit concepts/feature models
Featuring Reuse-driven Software Engineering Business (FeatuRSEB)

- Concurrent model building process

```
Exemplars

Domain Experts

Early domain models

Requirements

Use case model

Sources

Concurrent modeling cycle

Domain scoping, context modeling

Feature Model
```
Organization Domain Modeling (ODM)

- Sharing of concept and feature models across domains
  - Feature
  - Concept model