OBSE – an approach to Ontology-based Software Engineering in the practice

EMISA Workshop, St. Goar, 8./9. Oktober 2007
Andrej Bachmann, Wolfgang Hesse, Aaron Ruß (Univ. Marburg)
Christian Kop, Heinrich C. Mayr, Jürgen Vöhringer (Univ. Klagenfurt)

- What does OBSE mean?
- Goals of our project
- Ontology vs. software development cycles and their connections
- The role of glossaries in the OBSE cycle
- Tool support for OBSE
- An architecture for an OBSE tool
What does OBSE mean?

Definition 1:

An ontology is an explicit specification of a conceptualisation (T. Gruber [Gru 95]).

- Conceptualisations are not only needed for robots and agents but also for human designers – e.g. of database or application domains of software projects.
- Software developers normally derive the needed domain knowledge from requirements – sometimes from models of previous projects.
- Analysis and design for software projects might very well profit from existing ontologies covering their application domain.

Definition 2:

Ontology-based Software Engineering (OBSE) is a new approach for integrating ontologies in the SE process. Ontologies are used as reservoirs of domain knowledge for starting software projects and – vice versa – can gather domain knowledge from concluded projects.
Sources of domain knowledge in the SE process

- Domain knowledge base (ontology)
- Project requirements [NL]
  - Extract
  - Import
  - Transform & develop
  - Model (e.g. [UML])
  - Implement
  - Code (e.g. [Java])
Main questions

(1) What is the appropriate form (and language) for expressing ontologies in the SE context?

(2) How can the SE process be extended to an OBSE process including the use and evolution of ontologies?

Our answers (at a glimpse)

ad (1): For software projects in their early phases, semi-formal ontologies are well suited – e.g. in the form of glossaries.

ad (2): OBSE should follow an evolutionary approach – i.e. both software systems and ontologies (co-) evolve in a multi-cyclic, incremental process.
Goals of the project

- To develop and present a *new approach* to OBSE,
- to define *software development and ontology development* processes for co-evolution,
- to incorporate *glossaries* in the processes using the *KCP Methodology*\(^1\),
- to use an *evolutionary, multi-cyclic process model* appropriate for both processes and their combination: the (Marburg based) *EOS*\(^2\) approach,
- to build *tools* supporting the OBSE process(es).

\(^1\) *KCPM: Klagenfurt Conceptual Predesign Methodology*

\(^2\) *EOS: Evolutionary, Object-oriented Software development*
Combining the life cycles

Ontology life cycle

- Domain knowledge (NL)
  - extract
  - transform
  - revise
- Ontology (glossary form)
  - exchange knowledge
- Ontology (UML form)
- Ontology (OL form)

Software project life cycle

- Project requirements (NL)
  - extract
  - build
  - revise
- Project KB (glossary form)
  - transform
- System model (UML form)
- System version (PL form)
KCPM: *Klagenfurt Conceptual Predesign Model* (1)

**Problem:**

- End users and developers normally do not „speak the same language:"
- Users prefer natural language but NL-formulated requirements are often ambiguous and incomplete.
- Developers' conceptual models are not well understood by end users.
KCPM: Klagenfurt Conceptual Predesign Model (2)

**Problem:**
- End users and developers normally do not "speak the same language:
- Users prefer natural language but NL-formulated requirements are often ambiguous and incomplete.
- Developers' conceptual models are not well understood by end users.

**Solution:**
- Add a new intermediate step: *conceptual predesign*
- KCPM has only few orthogonal and intuitively understandable modeling concepts.
- KCPM is based on glossaries and mainly used for requirements elicitation, analysis and as a basis for validation.
Use of **KCPM** for managing domain ontologies

- Domain ontologies may be represented and managed by means of **glossaries**.

- They consist of
  - thing types,
  - connection types,
  - operation / cooperation types,
  - conditions,
  - …

- Forms of presentation:
  - as **table**,
  - graphical (as network)
  - **UML-like**

<table>
<thead>
<tr>
<th>c-id#</th>
<th>name</th>
<th>connection-type determiner</th>
<th>p-id#</th>
<th>involved thing-type</th>
<th>name</th>
<th>min/max</th>
<th>perspective determiner</th>
<th>perspective</th>
<th>requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>C001</td>
<td>author/book</td>
<td>book_writing</td>
<td>p001-1</td>
<td>D001, author</td>
<td>write</td>
<td>0..n</td>
<td>D007 writer</td>
<td></td>
<td>S1, S2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p001-2</td>
<td>D002, book</td>
<td>is_written_by</td>
<td>0..n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C002</td>
<td>identification</td>
<td></td>
<td>p002-1</td>
<td>D004, ISBN</td>
<td>identifies</td>
<td>1..1</td>
<td></td>
<td></td>
<td>j</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p002-1</td>
<td>D002, book</td>
<td>is_identified_by</td>
<td>1..1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C003</td>
<td>contract</td>
<td></td>
<td>p003-1</td>
<td>D005, publisher</td>
<td>contracts</td>
<td>0..n</td>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p003-2</td>
<td>D004, book_writing</td>
<td>is_contracted_by</td>
<td>1..1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**KCPM** network view on an ontology
KCP metamodel
EOS: *Evolutionary, Object-oriented Software development*

- Every component has its own process (= development cycle)
- All cycles are of similar shape, together they form a "fractal" structure
Ontology and Software life cycles intertwined
Bridges

- Central goal of the envisaged OBSE tool: to build *bridges* between the development paths.

- **Main bridges** are:
  - *import* bridges, e.g. between *domain ontology OU phase* and *software project SA phase*.
  - *export* bridges, e.g. between *software project SO phase* and *domain ontology OA phase* (ontology revision cycle).
The OBSE tool

- supports OBSE work
- main *target group*: SW developers who work on projects with ontology support
- offers *bridges* to support ontology editors:
  *Import*: from ontology to OBSE (analysis) tool
  *Export*: from OBSE tool to ontology editor
- offers *conversion* functions between KCP glossaries and UML class diagrams
- provides an *interface* to schema integration functions
The OBSE tool prototype

- builds on an eclipse-based implementation of **RCP (Rich Client Platform)**
- offers a **uniform interface** (e.g., for static-/dynamic editors)
- enables switching between different **views** (tables / graphics) using MVC techniques
- offers **synchronisation** mechanisms and transaction handling
- is **meta model-based**: database structure, editors and transformations (e.g. between KCPM and UML) are derived from KCPM meta model
Eclipse Plug-in Architecture

- The architecture provides a flexible way to build the OBSE tool: components of the tool can be created, e.g. by different teams in different projects
- It is based on **Plug-ins, Extension Points** and **Extension**
  
  **Plug-in**: component with a set of contributions
  
  **Extension Point**: a named entry for a collection of contributions
  
  **Extension**: integration of a contribution

- **RCP Platform** provides basic functionality and **Extension Points** for an Eclipse-based Application

- **Runtime** is a small layer that executes RCP applications on different platforms like Linux or Windows

(Source: www.eclipse.org/platform)
An architecture for the OBSE tool

RCP: Rich Client Platform

EMF: Eclipse Modelling Framework

GEF: Graphical Editing Framework

EPF: Eclipse Process Framework
- CIM's are (not) yet well defined in the MDA world.
- Glossaries might be well used as CIM's.
- MDA idea of continuous model transformations may be extended to the early phases by Glossary-to-UML (and vice versa) transformations.
Summary

• **Ontology-based Software Engineering (OBSE)** is a new approach for combining ontology and software project development processes.

• Software projects acquire domain knowledge not only from requirements but also from ontologies (= condensed knowledge from previous work).

• Vice versa, ontologies may profit from domain knowledge gained in (concluded) projects.

• Knowledge bases are glossaries – as defined by the KCP method.

• Development cycles are uniform and follow the EOS process model.

• Knowledge is transferred via import and export bridges linking the two kinds of cycles.

• Bridge functions are part of the OBSE tool – now operational on the prototype level.
Please remember:

3rd AIS SIGSAND-Europe symposium on IS analysis and design
in Marburg/Lahn
June 12-13, 2008
sponsored by GI / EMISA
Deadline for papers: January 15th, 2008
References


References (cont'd)


## General comparison

<table>
<thead>
<tr>
<th>Software Engineering Process</th>
<th>Ontology Engineering Process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Duration of process</strong></td>
<td><strong>Duration of process</strong></td>
</tr>
<tr>
<td>- fixed, until project</td>
<td>- not fixed, unlimited</td>
</tr>
<tr>
<td>termination</td>
<td></td>
</tr>
<tr>
<td><strong>Structure of process</strong></td>
<td><strong>Structure of process</strong></td>
</tr>
<tr>
<td>- Phases, maybe decomposed</td>
<td>- preferably iterations,</td>
</tr>
<tr>
<td>in</td>
<td>maybe decomposed in</td>
</tr>
<tr>
<td>iterations (e.g. RUP)</td>
<td>phases</td>
</tr>
<tr>
<td>- Activities: parts of</td>
<td>- Activities: mostly</td>
</tr>
<tr>
<td>phases</td>
<td>extensions, modifications</td>
</tr>
<tr>
<td><strong>Subprocesses</strong></td>
<td><strong>Subprocesses</strong></td>
</tr>
<tr>
<td>concern the development of</td>
<td>concern the development or</td>
</tr>
<tr>
<td>single components or</td>
<td>revision of partial areas</td>
</tr>
<tr>
<td>increments</td>
<td></td>
</tr>
<tr>
<td><strong>Process paradigms:</strong></td>
<td><strong>Process paradigms:</strong></td>
</tr>
<tr>
<td>- Waterfall</td>
<td>- incremental</td>
</tr>
<tr>
<td>- incremental</td>
<td>- component-based</td>
</tr>
<tr>
<td>- component-based</td>
<td>- prototyping, spiral</td>
</tr>
<tr>
<td>- prototyping, spiral</td>
<td>- evolutionary</td>
</tr>
<tr>
<td>- evolutionary</td>
<td></td>
</tr>
</tbody>
</table>