Ontological Analysis of KAOS Using Separation of Reference

Raimundas Matulevičius, Patrick Heymans
University of Namur, Belgium

Andreas L. Opdahl
University of Bergen, Norway
Outline

- Introduction
- KAOS
- Research method
  - KAOS abstract syntax
  - UEML approach
- Results
  - Example – KAOS Goal
  - Other KAOS constructs
- Discussion
  - Evaluation of UEML approach
- Conclusions and Future work
Introduction

- Goal-oriented approaches
  - What the new system should do?
  - How the system should be built?
- KAOS – Knowledge Acquisition in autOmated Specification
  - Support for representing, reasoning about and specifying trust during analysis and specification, but it offers less support for tracing and realising trust concerns during design and system generation.
- Ontological analysis of KAOS using separation of reference
KAOS

- Four models:
  - Goal model
  - Object model
  - Agent model
  - Operation model

- Constructs:
  - Graphical and a textual syntax
  - Defined using the real-time temporal logic

The 11th CAiSE’06 International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD’06), Luxembourg
Research method

- Definition of KAOS abstract syntax
  - Identify language constructs;
- Application of the UEML approach to the KAOS constructs
  - Define meaning of each construct
KAOS abstract syntax

- "Reasoning about Agents in Goal-oriented Requirements Engineering" [Letier, 2001]
  - Mathematical definition of abstract syntax and some semantics
  - Precise, but partial
  - Intertwined with other topics
- "The KAOS meta-model - Ten years after" (technical report) [van Lamsweerde, 2003]
  - Meta-model of all KAOS models
    - Abstract syntax, concrete syntax, informal semantics
  - Imprecise
    - Meta-modelling language is not standard
    - Missing multiplicities, specialisation constraints, abstract classes and integrity rules
KAOS abstract syntax (cont.)

Limited to the **goal** model
BWW and UEML approach

- The UEML approach uses the BWW model to describe modelling constructs:
  - Maps a modelling construct onto a specific ontological class, property, state or event.
  - Modelling construct may represent a scene where multiple classes, properties etc. play parts.
UEML approach

**Preamble**
1. Construct name
2. Alternative construct names
3. Which language the construct is part of
4. Language acronym and references
5. Which diagram types the construct is used in
6. Diagram type acronym and references

**Presentation**
1. Icon/line style user-definable attributes
2. Relationships to other constructs
3. Cardinality restrictions
4. Layout conventions
UEML approach (cont.)

❖ **Representation,**
what modelling construct intends to represent:

- Instantiation level
- Class (-es)
- Property (-ies)
- Behaviour
  - *Existence* of classes, things and properties? (static)
  - *States, transformations, processes* (dynamic)
- Modality
  - permission, obligation, recommendation, etc.
UEML approach (cont.)

- **Construct description for each language construct:**
  - Preamble
  - Presentation
  - Representation

- All the modelling constructs in the UEML are thereby *interrelated at the most detailed level possible* via the common ontology.
Mapping of KAOS Goal (cont.)

The 11th CAiSE’06 International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD’06), Luxembourg
Mapping of KAOS Goal (cont.)

- **Maintain and Avoid**
  - theGoal represents BWW-\textit{stateLaw}

- **Achieve and Cease**
  - theGoal represents BWW- \textit{transformationLaw}

- **Ability to be**
  - assigned -- \textit{Assignement};
  - refined -- \textit{G-refinement};
  - operationalised -- \textit{Operationalisation};
  - conflicting -- \textit{Conflict}. 
### KAOS constructs

<table>
<thead>
<tr>
<th>Construct</th>
<th>Inst</th>
<th>Primary Class/Property</th>
<th>Behaviour</th>
<th>Modality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieve goal</td>
<td>Both</td>
<td>Transformation law</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Agent</td>
<td>Both</td>
<td>Active component thing</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Assignment</td>
<td>Both</td>
<td>Complex mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Avoid goal</td>
<td>Both</td>
<td>State law</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Boundary condition</td>
<td>Both</td>
<td>State law</td>
<td>State</td>
<td>Fact</td>
</tr>
<tr>
<td>Cease goal</td>
<td>Both</td>
<td>Transformation law</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Conflict</td>
<td>Both</td>
<td>Mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Control</td>
<td>Both</td>
<td>Binding mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Domain property</td>
<td>Both</td>
<td>Any property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Environment agent</td>
<td>Both</td>
<td>Active component thing</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Event</td>
<td>Type</td>
<td>Changing thing</td>
<td>Event</td>
<td>Fact</td>
</tr>
<tr>
<td>Expectation</td>
<td>Both</td>
<td>Complex law property</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Goal</td>
<td>Both</td>
<td>Complex law property</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>G-refinement</td>
<td>Both</td>
<td>Complex mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Input</td>
<td>Type</td>
<td>Binding mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Maintain goal</td>
<td>Both</td>
<td>State law</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Monitor</td>
<td>Both</td>
<td>Binding mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Object</td>
<td>Both</td>
<td>Component thing</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Operation</td>
<td>Type</td>
<td>Transformation law</td>
<td>Process</td>
<td>Fact</td>
</tr>
<tr>
<td>Operationalisation</td>
<td>Both</td>
<td>Complex mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Output</td>
<td>Type</td>
<td>Binding mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Performance</td>
<td>Type</td>
<td>Complex mutual property</td>
<td>Existence</td>
<td>Fact</td>
</tr>
<tr>
<td>Requirement</td>
<td>Both</td>
<td>Complex law property</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Softgoal</td>
<td>Both</td>
<td>Law property</td>
<td>Existence</td>
<td>Intent</td>
</tr>
<tr>
<td>Software agent</td>
<td>Both</td>
<td>Component software thing</td>
<td>Existence</td>
<td>Fact</td>
</tr>
</tbody>
</table>
Discussion

- Fragmentation of research [Kavakli and Loucopoulos, 2005].
  - "Usage" what RE activity does the GOA support?
    - each GOA tends to focus on one RE activity
  - "Subject" - what is the nature of goal?
    - different definitions and categories of goal
  - "Representation" - how are goals expressed?
    - different abstract and concrete syntax of goals
  - "Development" - how are goal models developed?
    - different methodological and tool support

- We focus on the "subject" and "representation" suggesting possibilities for model transformation and the language integration.
Discussion (cont.)

- Model transformation

  - From GRL to KAOS
  - From KAOS to GRL

- Language limitations:
  - Referential redundancy;
  - Referential overload;
  - Ontological incompleteness;
  - Referential under-definition (excess)
Evaluation of the UEML approach

Limitations:

- The UEML approach is difficult to use because it is based on a particular way of thinking.
- It is hard to
  - determine exactly which language part constitutes a modelling construct;
  - find the appropriate classes, properties, states and events in the common ontology to use when describing a construct;
  - judge when to choose an existing class, property, state or event in the ontology and when to define a new one.
Evaluation of the UEML approach (cont.)

Advantages:

- The UEML approach offers:
  - a detailed advice on how to proceed when analysing individual language constructs;
  - construct description at a high level of detail, which tends to integrate languages at a fine-grained, precise level which leads to complete and easily comparable construct descriptions;
  - ontological analysis in terms of particular classes, properties, states and events, and not just in terms of the concepts in general;

- It has a positive externality
  - each construct becomes easier to incorporate as more constructs are already added to the UEML;
  - each language becomes easier to incorporate as more languages are already added.
Tool support

UEMLBase
Conclusions and Future Work

Languages compared using the UEML approach
- Translate models based on their semantics
- Integrate languages and models

Future work:
- Analysis of other goal-oriented languages.
- Application of the UEMLBase to analyse other languages
- Negotiate language evaluations with partners