An Ontology-based Modeling Tool for Knowledge-intensive Services

ATHENE

Simon Nikles, June 2007
Problem of semantic languages

Semantic languages offer expressive power

Business people and domain experts

- lack the experience of semantic technologies
- should not be forced to deal with ontologies or rule syntaxes
Goals and Ideas I

- Application-specific ontology development

- Build any type of model (user defined)

- Extensibility: Define any model-type (process, organization etc.) and its graphical representation in one tool

- Combine different models, e.g. process-model and organization-model

Reuse and export models

- e.g. export to BPEL or map to OWL-S
Goals and Ideas II

Maintain and query a repository of models

• Find processes by attributes

• Find processes by used services

• Find services used in a process

• Find dependent or related processes

• Find processes related or dependent to specific organizational-resources

…
Requirements I

Different levels/interfaces for users are needed

• To define model-types

• To use the model types for creating new models (processes etc.)

an expressive language is needed

• To store and use additional information which is needed e.g. to export

• To find processes and query their attributes and relations

• To create and combine ambitious models
Solution approach

Users don’t need knowledge of semantic languages, because…

With ATHENE we enable users to

- **model** processes etc. in a known graphical manner
- assign **additional information** through attributes (e.g. needed for target-formats)
- **store the model** with its information as an ontology (OWL)
Defining new models requires an adequate modelling level

Two levels are visible for the user:

On **meta-level** model-types are defined (Objects / Relations, graphical representation, attributes / restrictions)

On the **modelling-level**, the defined objects are available to create models
Defining new models requires an adequate modelling level.

Two levels are visible for the user:

On **meta-level** model-types are defined (Objects / Relations, graphical representation, attributes / restrictions).

On **modelling-level**, the defined objects are available to create models:

- Atomic-service, composite service
- Sequence, condition, switch
- Iterate, while
- …

Check form
if form correct
Confirm receipt
Save application

…”
Meta-meta-approach (I)

While the user has an interface / workspace for the mentioned levels ATHENE works with a 3-level-model whereas each level allocates classes for the level beneath. The meta²-model is expressed explicitly as OWL definition.

meta²-Model (meta-meta-level)

On this level, ATHENE provides a basic set of classes which are available for a model-type

This set allows to define modeling-objects. Particularly objects, relations and attributes of each object/relation can be defined
Meta-meta-approach (II)

meta-model

• on the meta level users can specify model-types

• Model types consist of subclasses of the meta-2 classes

model

• On modeling level the user can actually create domain specific models as processes or organizational models

• On this level the classes of the meta-level are instantiated
OWL Representation

Each model is stored in owl and represents a consistent self-contained OWL-model through the 3 Levels.

The implementation works not directly on OWL, it loads the models into an internal class-model:

• Changing the meta²-model could lead to inconsistencies of meta-models and models, so it should be constant anyway

• The program needs knowledge of the meta²-model for programmatic operations. Adding these control- and representation information to the model, would blow up the model, without having real advantages.
Meta-level
Modelling-level
Implementation

ATHENE is realized as java applet with a plug-in concept.

- The webaccess allows to share a common repository of model-types and models without exchanging files.

- Except the browser and an up-to-date virtual machine, no software has to be installed.

- Components (meta-modelling and modeling by now) are realized as plug-ins, independent from each other. The system therefore can be extended easily.