Meta Modelling Tasks – Prototypical Language Features

Jürgen Jung and Lutz Kirchner
Chair of Information Systems and Enterprise Modelling
University of Duisburg-Essen
45141 Essen, Germany

This document contains several small tasks. Each of these tasks represents a prototypical feature of modelling languages. The problems given in the tasks should be implemented using a meta-modelling tool. The document at hand is structured as follows:

It is not important to implement all these language features. Existing modelling language implementations should be reused. The focus is on showing how the tasks can be solved by using a given meta-modelling tool.

1 Static Structure of Diagrams

The static structure includes language features like entity types, relationship types, attributes and constraints on static elements.

a) Attributes

Description: Properties of static elements have to be represented. Such properties encompass its name and a set of attributes. An attribute consists at least of a locally unique name and a type description.

Task: Implement a model element representing a UML class with attributes. Every attribute has a locally unique name, a type and cardinality. Optionally, the visibility (public, private, protected) of an attribute can be specified by a graphical symbol.

Example: A class in UML consists of a unique name and encapsulates attributes and methods. Attributes are displayed inside the middle part of a class symbol (cf. Figure 1). Class C has two attributes attrA1 and attrA2 with types A and B respectively. The cardinality of the first attribute is set to 'exactly one' and the one of the second attribute to 'one or many'.

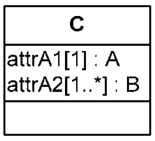

Figure 1: A UML class with two attributes

b) *Relationships*

Description: Binary relationships between static elements have to be represented. Every relationship has a name and consists of two relationship ends. A relationship end has a role qualifier and a cardinality qualifier. A role connects one static element to the end of a relationship.

Task: Implement a model element representing a UML binary association with the features given.

Example: An association in UML connects two classes (cf. Figure 2). Such an association has a locally unique name and (if required) an association class – representing properties of an association.

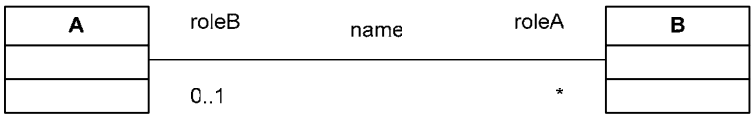

Figure 2: Association

c) *Constraints between relationships*

Description: Constraints between relationships address aspects such as whether a relationship requires another (implication), excludes another (XOR) and similar aspects. Possible constraints are listed below:

- AND: All relationships are required
- OR: one or more relationship might be established
- XOR: only one relationship might be established
- SUBSET: the set of instances of one relationship is a (true) subset of the set of instances of the other relationship
- Implication: If relationship A is established also relationship has to be established, too.

Task: Implement a diagram allowing the specification of constraints between relationships. Consistency and plausibility check have to be provided, too.

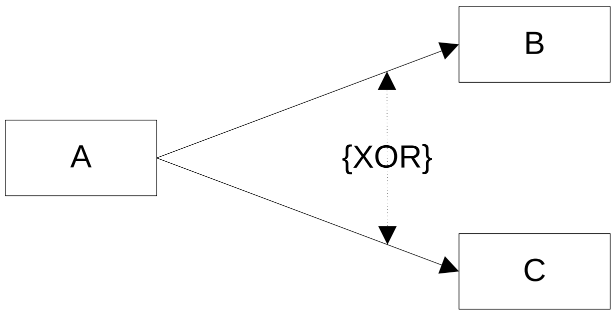

Figure 3: Example constraint

Example: An example for a constraint between two relationships is shown in Figure 3. An instance of type A has either a link to an instance of type B or type C. It has never a link instance of both types.

d) Integrity constraints

Description: Diagram properties address integrity constraints on specific diagram types.
Task: Implement a diagram type for modelling organisational charts. This diagram type should contain organisational units and super/subunit-relationships. Checks for forbidden relationships (i.e. cycles) have to be implemented, too.
Example: An organisational chart displays the static structure of an organisation (an example is given in Figure 4). Organisational units are represented by rectangles and lines from the bottom of a rectangle to the top of another rectangle. This construct represents superunit-subunit-relationships between organisational units. A, for example, is a superunit of A1 and A2. A21 and A22 are subunits of A2. But, A22 must never be a superunit of one of its direct or indirect (i.e. transitive) superunits.

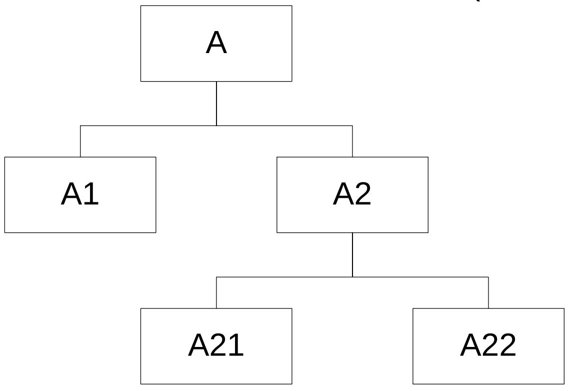

Figure 4: Example Organisational Chart

Comparable Features: Checks for dedicated graph properties and/or types (like tree or DAG))

e) Bipartite graphs

Description: A special kind of graph is a bipartite graph. A bipartite graph is a graph with the following constraint: There are two types of nodes which appear alternating in a diagram.
Task: Implement a diagram editor for Petri nets. The constraint of alternating nodes has to be checked.
Example: Petri nets are bipartite graphs consisting of two types of nodes: places and transitions. Every place is followed by none, one or several transitions and every transition is followed by zero or more places. No place is followed by another place

as well as no transition has another transition as successor. A simple example of a Petri net is given in Figure 5.

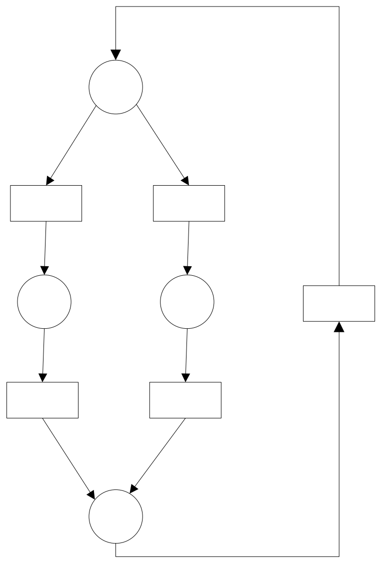

Figure 5: Example Petri net

f) Well-formed graphs

Description: A graph is said to be well-formed if there is a corresponding synchronisation to every branch in the control flow and there is no path starting from the branch which is not reaching the synchronisation.
Task: Implement a constraint checking mechanism which tests for well-formedness on demand.
Example: In MEMO-OrgML every parallel branch results in a synchronisation (cf. Figure 7).

2 Visualisation of nodes and relationships

Visualisation addresses the graphical representation of diagram elements. Usually, the semantics of a model element is bound to a special graphical notation.

a) Geometrical figures

Description: Different types of nodes have to be displayed by different geometrical symbols.
Task: List all geometrical figures available for the definition of nodes in diagram. Demonstrate the construction of complex model elements using geometrical figures.
Example: ER-Diagrams use rectangles and diamonds; UML class diagrams have nodes consisting of three composed rectangles -- one of them for each of a class' aspect: general part (name, stereotype, …), attribute and operations.

b) Complex graphics

Description: Different types of nodes have to be displayed by complex symbols.
Task: Use scalable complex images (i.e. vector based images) for the representation of nodes.
Example: MEMO-OrgML has different visual representations for processes and events (cf. Figure 8 and Figure 7). MEMO-OrgML (MEMO Organisation Modelling Language) is a language designed for modelling organisations. MEMO (Multiperspective Enterprise Modelling) is a method for enterprise modelling.

c) *Complex graphics at relationship ends*

Description: Different types of arrows have to be displayed by complex symbols. This may be different arrow types or other geometrical shapes like circles, rectangles, ellipses or any combination of those.
Task: Demonstrate how to use a scalable graphics format for the representation of relationships. Show that the size of a complex arrow type can be changed in order to proportionally fit the size of the attached node element.
Example: MEMO-OrgML uses a special symbol for the representation of processes' decomposition (cf. Figure 6).

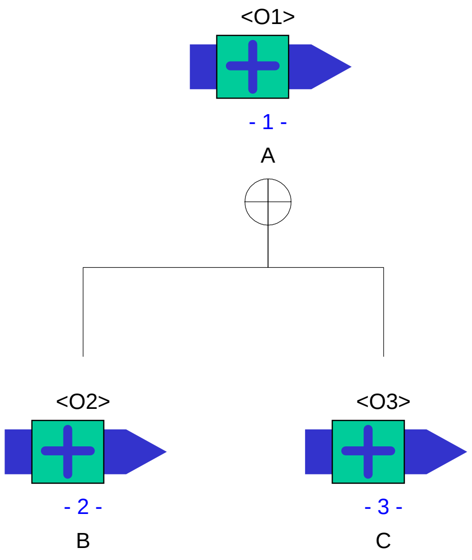

Figure 6: Example process decomposition (MEMO-OrgML)

d) *Semantics by different routing of lines*

Description: Different types of relationships should be displayed by using different line routing in the diagram.
Task: Implement a simple process diagram with parallel and alternative control flow as specified by the MEMO-OrgML.
Example: Control flow in the MEMO-OrgML process modelling language is displayed using two types of edges. Parallel control flow is modelled using arrows consisting of horizontal and vertical elements (cf. Figure 7). In contrast to this, an alternative is annotated by straight lines (cf. Figure 8).

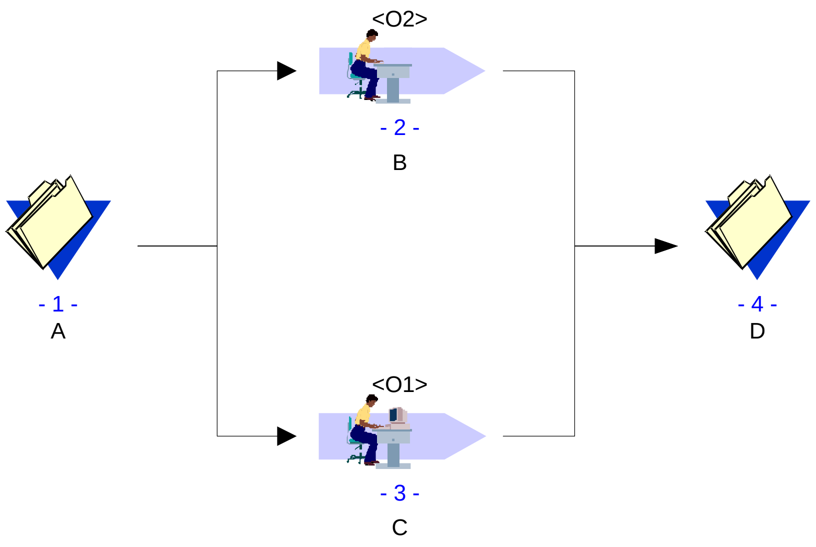

Figure 7: Parallel Execution of Processes B and C (MEMO-OrgML)

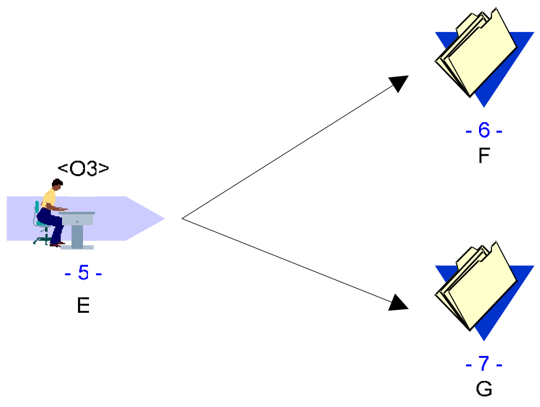

Figure 8: Alternative control flow after process E (MEMO-OrgML)

3 References

a) Links between a model element and another model

Description: Usually, a model element can be further specified by a different model. Hence, a meta-modelling tool has to manage references between model elements in a given model and other models.
Task: Demonstrate the mechanism for describing links between model elements and other models given in your meta-modelling tool.
Example: In UML a class (represented in a class diagram) can be further specified by a state chart. Or: A data store in a data flow diagram (DFD) can be specified by an entity relationship diagram (ERD). Or: Tuples on places in a Predicate/Transition-Net (a special kind of higher Petri nets) can be described by predicates.

b) Links between a part of a model element and another model

Description: Parts of model elements can sometimes be specified by a different model. Hence, a meta-modelling tool has to manage references between model elements' parts in a given model and other models.

Task: Describe the mechanism for specifying links between a model element's part and other models given in your meta-modelling tool.
Example: In UML the operation of a class can be further specified by an interaction diagram describing the behaviour of the operation.

c) *External links*

Description: Usually, model elements can be further specified by external
- documents
- applications.

Task: Name and explain mechanisms for the description of model elements and external files and applications. Which of the following types are possible for external sources:
- File
    - Remote document
    - Local document
- Application
    - Web service
    - Remote application
    - Local application

Example: The documentation of a class can be kept in an external Word-document. A symbol representing a component in UML might be associated with an executable file. The DTD of a document might be associated with an external XML-document. The operation of a UML class can be implemented by using a web service.

d) *Decomposition of a model element*

Description: Some model elements like business processes may be composed of other business processes. To depict this composition a meta-modelling tool has to link a diagram showing subprocesses of the superordinated process in a way that the balance of incoming and outgoing flows is kept. In other words the target and source of the control flows that go in and out of the superordinated model element remain the same when used in a decomposition diagram of that element. Additional support of hierarchical numbering of the model elements would be appreciated (see Figure 10).
Task: Demonstrate the mechanisms provided by the tool that support the creation of balanced decompositions. Explain whether the mechanisms are preventive (i.e. preventing the creation of unbalanced diagrams) or reactive (highlighting possible inconsistencies after creation).
Example: A business process, e.g. *Customer Support* (see Figure 10), may be composed of other business processes like *General Support*, *Technical Support* among others (see Figure 10). *Customer Support* has an incoming as well as an outgoing control flow to and from events. The control flows, there targets and goals must be compatible to the ones used in any decomposition of the superordinated process.

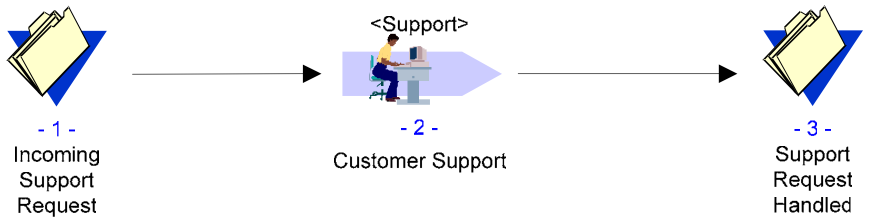

Figure 9: Example Process *Customer Support* (MEMO-OrgML)

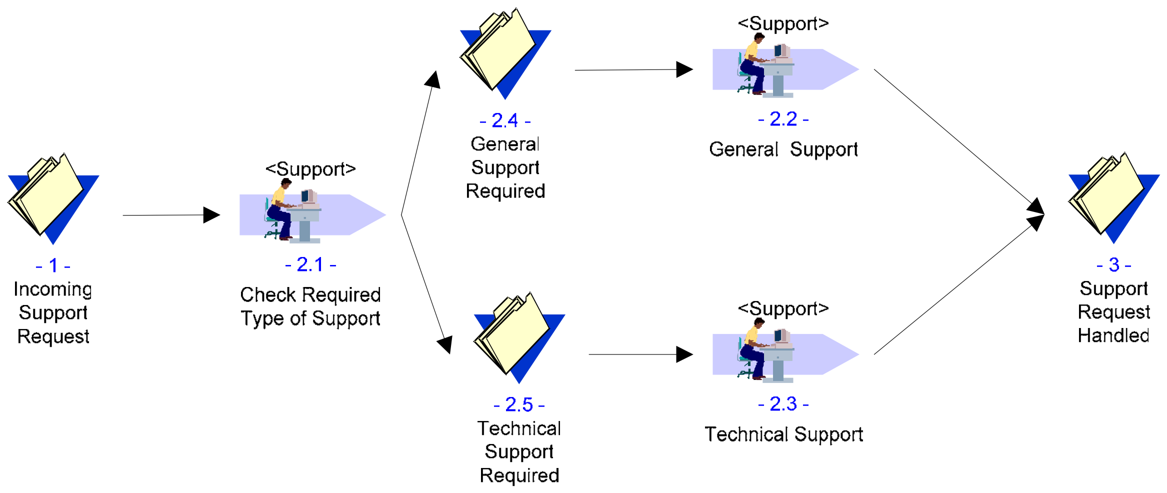

Figure 10: Decomposition of *Customer Support* (MEMO-OrgML)

4 Further Aspects

In addition to the concepts described above, there are other concepts or specific features, a tool vendor may want to point to. Examples include:

- dynamic aspects (i.e. Simulation)
- code generation
- re-engineering (e.g. construction of models basing on given code)
- report generation
- user interfaces
- integration with existing software
- version management
- extensibility