MAPPING FUNCTIONAL REQUIREMENTS: FROM NATURAL LANGUAGE TO CONCEPTUAL SCHEMATA

Christian Kop, Heinrich C. Mayr
University of Klagenfurt
[chris|mayr]@ifit.uni-klu.ac.at

ABSTRACT

Natural language requirements specifications for complex application systems are difficult to analyze and to validate. Consequently, a more formal language for requirements representation is needed that is, for the purpose of validation, understood by end-users (i.e. the requirements owners), as well as by system analysts and developers. Often, the Unified Modeling Language UML is proposed for that purpose. But it is the authors' experience that especially in non-technical environments UML does not meet the capabilities and interests of end-users so that many development projects still suffer from an incomplete requirements validation. We therefore propose an approach that is based on a more end-user oriented modeling language and that provides mechanisms for both: an automatic transformation of natural language requirements specifications into that modeling language, and after validation, an automatic transformation to UML or other conceptual languages. This paper concentrates on how our approach deals with requirements concerning information system behavior.

KEYWORDS

Behavior Modeling, *Conceptual Modeling, Dynamic Modeling, Requirements Engineering*

1. INTRODUCTION

Usually, information system development projects start with textual requirements descriptions and transform these into a conceptual schema based on a particular conceptual model (e.g., using the UML). For analyzing behavioral aspects on the natural language level, so-called scenarios are very popular (see e.g. [14]). In practice, these are used for different purposes (see, e.g. [18]). [10, 13, 19] propose specific scenario construction processes to overcome problems in managing, refining and integrating textual scenarios. The subsequent transformation of textual scenarios into graphical notations like sequence diagrams, event traces or state charts, is examined, e.g., in [15] and [13], the latter source proposing the exploitation of linguistic verb features. Another approach [17] specifies a rather detailed construction and refinement process for sequence diagrams. [15] and others introduced mechanisms for deriving state charts from sequence diagrams. [8] discusses the coupling of use cases with class models based on so-called activity charts. Despite of all their merits, these approaches come with the following disadvantages:

(1) Information system projects often suffer from incomplete or inadequate requirements specifications even if extensive conceptual modeling is carried out. In our opinion, these problems result from the fact, that traditional conceptual models are too abstract as to be easily understood and thus thoroughly validated by average users, who are the stake-holders of the relevant concepts, notions and requirements. This is especially true for non-technical environments (for a more detailed discussion see [9] and [12]).

(2) Sequence diagrams are very appropriate means for discussing and clarifying processes (in the sense of event flows) from a global use case oriented view. However, the local framework of pre- and side-conditions that drive an actor's contribution to such a process is spread over several (possibly many) variants of sequence diagrams which hardly can be overseen and judged by the person who is responsible for that particular contribution.

(3) In real world information system projects hundreds and even thousands of sequence diagrams would have to be produced for a complete documentation of all (functional) behavioral aspects. This becomes even more cumbersome by the fact that requirements elicitation and analysis is an iterative task which leads to repeated changes and thus re-writings or re-drawings.

We therefore propose a lean modeling language that (1) supports an end-user oriented view, (2) allows for expressing as much information as possible gathered from textual requirements specifications so that recurring to the textual source is not necessary in subsequent steps, (3) used as an inter-lingua enhances automatic transformation from the textual sources and to the conceptual schema. We call that approach *Conceptual Predesign (CP)* since it is based on a semantic (meta)model and since it precedes the usual conceptual design. The model used (*KCPM: Klagenfurt Conceptual Predesign Model*) offers a set of semantic concepts for modeling static and dynamic aspects of a Universe of Discourse (UoD). It is based on an ontology which treats UoD´s as systems consisting of interrelated elements (things) that are able to perform services (operations) and that are activated by events

(messages sent by other things). KCPM Schemas are represented in either glossary like or diagrammatic form. Within this paper we restrict on the latter, for the more end-user oriented glossary version see, e.g. [12]. KCPM may be seen as an extension of DATA-ID [3]. The Color-X approach [2] works similarly with an inter-lingua, however focuses on gathering events.

In what follows we first introduce the main KCPM concepts for behavior modeling (sect. 2). We then discuss how these are associated to natural language phrases (sect. 3). Using activity diagrams as provided by UML as an example, sect. 4 outlines the mapping of KCPM schema entries to equivalent conceptual constructs. Sect. 5 summarizes the presented work and outlines how other dynamic models can be derived from our inter-lingua.

2. KCPM NOTIONS FOR BEHAVIOR MODELING

Comparing the different approaches to conceptual behavior modeling (e.g. *state charts*, *sequence diagrams*, *use cases*, *activity diagrams* etc.), there seems to be a small set of basic principles:

- there are *actors* capable to execute *tasks/services*,
- task execution is coupled with the manipulation of some objects (including message sending),
- the execution of a task depends on some pre-conditions,
- the execution of a task results in some post-conditions,
- objects are related to pre- and post conditions.

In the context of requirements elicitation and analysis for business information systems it seems to be useful to concentrate on these principles. Therefore, KCPM is restricted to two main concepts for behavior modeling: **operation type** and **co-operation type**.

Operation types are used to model functional services, that can be called via messages (service calls). As such they may bee seen as a generalization of the notions use case, activity, action, method, service etc. Each operation type is characterized by references to so-called **thing types**, which model the *actors* (executing instances of the resp. operation type), *service parameters* and the *receivers* of the service results. The notion of thing type is the central KCPM concept for structure modeling [9] and may be seen as a generalization of the usual conceptual notions *class* (or entity set) and *value type*[1]. Typical things (instances of thing types) are natural or juristic persons, material or immaterial objects, abstract notions. In textual requirements specifications they are usually referred to by noun phrases.

UoD dynamics emerge from *actors* (thing types) performing operations under certain circumstances (*pre-conditions*) and thus creating new circumstances (*post-conditions*). This is captured by the KCPM concept of **co-operation type**[2] which is formally (simplified) defined as a triple *<prc,{(A,O)},poc> where prc* and *poc* are sets of conditions (possibly combined by logic expressions), and {(*A,O*)} is a set of actor, operation type pairs. Figure 1 illustrates this definition: Co-operation types are represented by rectangles, operation types by ellipses. The latter are drawn into the rectangle of those co-operation types they are contributing to. Conditions are represented by filled circles. c is a pre- (post-) condition of co-operation type CT, if an arrow point from c to CT (CT to c). Each condition may be pre- and/or post-condition of one or more co-operation types. In this way, a network of co-operation types may be constructed. For each operation type and condition we depict the 'involved' thing types: parameters and actors<A>.

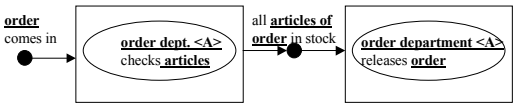

Figure 1: cooperation types, operation types, pre- and post conditions – a graphical notation

3. LINGUISTIC ASPECTS OF KCPM SCHEMA CONSTRUCTION

Co-operation types are derived from textual requirements specifications by an iterative sentence by sentence execution of the three following steps:

1) **Pattern selection**: a particular sentence/phrase is mapped to a condition, an operation type or a co-operation type primitive. We use the linguistic theory NTMS [11] for that purpose. NTMS is a generative language grammar that treats verb-phrases as heads of sentences and, like the case grammar [4], provides semantic roles for the natural arguments of verbs. Verbs are classified into 12 distinct verb classes that again may be used, for the purpose of pattern selection, to distinguish the following 5 phrase categories [5]:
   - *activity/action* (someone/something does somewhat): built by all kind of verb classes which have an actor role. Induces the selection of an operation type primitive.
   - *property/state* (the verb expresses a static relationship): e.g. *to **have***, *to **contain***, *to **be***. Induces the selection of a pre- or post-condition primitive.
   - *event* (ergative verbs describing primarily events instead of activities): e.g. *the deadline **expires**, the order **comes in***. This category also comprises mo-

[1] Value type corresponds to value set in the ER-model or to typed expression in UML. A KCPM value type may be user defined (e.g. Color, Customername etc.). It is not restricted to simple types (e.g. Number, Date etc.).

[2] This term has been taken from object orientated business process modeling [2]. A cooperation in that sense is an elementary step of a business process to which one or more actors contribute by (concurrently) executing operations (services).

dal verbs like *want*, *wishes* etc. Induces the selection of a pre-condition primitive.
- *completion of activity* (activity/action verbs in past participle form): e.g. *the order **is signed***. Induces the selection of a pre- or post-condition primitive.
- *restriction* (temporal or causal restrictions): e.g. *on Friday….* Induces the selection of a pre-condition primitive.

Co-operation type primitives are derived from more complex **if/then** constructions, e.g. from a combination of pre-condition and operation type (IF <prc> THEN <O>) or from a combination of operation-type and post-condition (IF <O> THEN <poc>> [5].

2) **Completion**: the primitive is complemented if necessary.

For instance, if an operation type primitive has been selected, where the actor is not referred to within the related activity phrase, or where some of the parameters are missing, then this actor/parameter has to be supplemented. Another example is the completion of a co-operation type primitive, where the post conditions are not yet known. Sometimes these may be derived from the related activity phrase (e.g. *IF <prc> THEN X releases order* implicitly contains the post-condition *order is released*).

3) **Integration**: the completed primitive is integrated into the co-operation type schema.

This step is supported by the construction guidelines depicted in figure 4. We start from a given primitive G_i and investigate the schema information available so far: S_1 corresponds to the situation where only a related precondition exists, S_2 to all other situations (i.e., there is already a related co-operation/operation type entry). Then, the matching case is selected from the table.

Treating these steps in detail would mean going far beyond the scope of this paper. We therefore restrict ourselves to illustrate the construction based on a simple example which is taken from [6] and slightly modified:

(1) The order comes in. (2) The order department checks each article on the order. (3) If each article is available on stock, then the order department relates it to the order. (4) Then, if the stock quantity of an article is lower than the minimal stock quantity, the stock clerk must order this article. (5) If the order comes in, also the bookkeeping department checks the payment. (6) If the payment is authorized and we have all articles in stock, then the order department releases the order. (7) If payment is authorized but there are not all articles in stock, then the order department must put the order to the pending orders. (8) If the payment is not authorized, then the order department must reject the order.

Sentence (1) is an event phrase due to the ergative verb "*comes in*". Therefore it is mapped to a pre-condition which forms the first schema element. Sentence (2) represents an activity phrase and is thus mapped to an operation type. We can now use construction guideline G_4/S_1 to complete the previously found schema element by forming a co-operation type. Sentence (3) is an *IF <prc> THEN O* construct to which we assign a co-operation type primitive consisting of a pre-condition and an operation type. We therefore may apply the construction guidelines of G_5/S_2. Since there are two alternatives (A, B), it is necessary to make a decision. Choosing alternative B means that we accept the pre-condition of the new schema element also to be the post-condition of an existing one. Sentence (4) is transformed and integrated similarly to sentence (3) using again guideline G_5/S_2 with alternative B. Sentence (5) leads to a co-operation type primitive, the pre-condition of which is matched according to G_5/S_1,B with the initially found one ("*order comes in*"). Thus, a new path within the co-operation type schema starts. From sentence (6) we can derive a co-operation type with two pre-conditions. We again may apply G_5/S_2,B to extend the schema. This process is continued exploiting sentences (7) and (8) an applying guideline G_5/S_2,B.

Usually textual requirements specifications do not contain "trivial" post-conditions explicitly (e.g. "the order is rejected" as a post-condition of rejecting). However, we might add such post-conditions for a more complete description. In particular, such post-condition might again be matched with pre-conditions of further co-operation types, thus contributing to a stronger connection of the schema components. The completed schema resulting from that process is presented in figure 2.

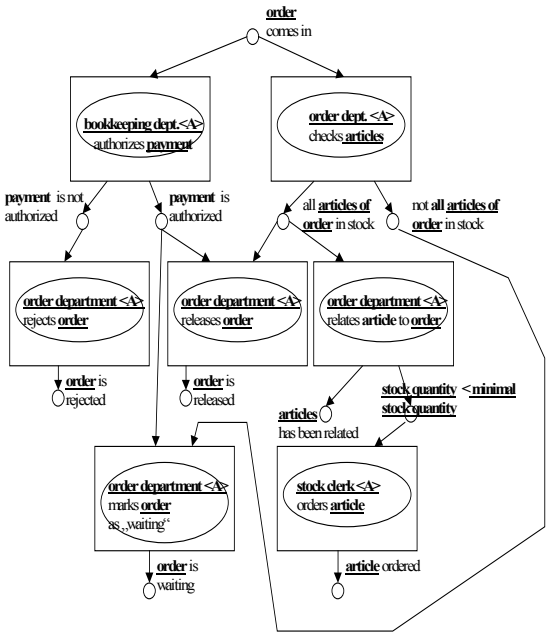

Figure 2: example cooperation type schema

4. MAPPING TO AN ACTIVITY DIAGRAM

A predesign schema as described above holds the essential information that is necessary to construct an initial schema for different conceptual models. E.g., mapping the co-operation type schema to an activity diagram is quite simple by the following heuristic rules:

- If a co-operation type contains only one operation type, then this operation type is mapped to an activity A_O. Each pre-condition is mapped to a transition to A_O. Each post-condition is mapped to a transition from A_O.
- If a co-operation type CT consists of a set of operation types, then the co-operation type is mapped itself to an activity A_{CT}. Each of its operation types O_1, O_2, .. O_n is mapped to a sub-activity A_{O1}, A_{O2}, ... , A_{On} within A_{CT}. At the beginning, a splitting is introduced, at the end a synchronization bar for all sub-activities. Each pre-condition is mapped to a transition to A_{CT}. Each post-condition is mapped to a transition from A_{CT}. If the pre-condition is derived from an event phrase, then the pre-condition is mapped to an UML-event on the transition. If the pre-condition or the post-condition does not represent an end-of activity phrase, then it is mapped to a conditional guard of the resp. transition.

Since we even know which thing types are involved into pre- and post-conditions we can also extend the mapping to generate an object flow from the co-operation type schema. Each pre-condition with a thing-type involved is a candidate for an object flow to an activity. Each post-condition with a thing-type involved is a candidate for an object flow from an activity. Moreover, there are hints for responsibilities swim lanes, e.g. an actor of an operation type is a candidate for a swim lane.

These heuristics support the mapping of single co-operation types. To generate the whole activity diagram, we must stick together the mapped concepts according to the whole co-operation type schema by matching activity diagram transitions based on the matching of their related pre-and post-conditions.

For illustration, let us again have a look at our example. Since there is no co-operation type with more than one operation type, all operation types are mapped to activities (*bookkeeping department* ***authorizes payment***; *order department* ***checks articles***; *order department* ***rejects order***; *order department* ***releases order***; *order department* ***relates article to order***; *order department* ***marks order as a "waiting"***; *stock clerk (additionally)* ***orders article***). Then, all pre- and post conditions are mapped to guarded conditions on the transitions between the activities. This results in an activity diagram as given in figure 3. In this activity diagram the synchronization and splitting bars are used in a somewhat artificial fashion. Nevertheless UML and Rational Rose[3] allow transitional relationships between splitting/synchronization bars. The splitting bars are used to show that from a given situation all subsequent activities can be executed independently. The synchronization bar is used to express that a certain activity depends on the previous execution of one or several other (independent) activities. This semantic is also true for the "*payment is authorized*" which points to two synchronization bars (a synchronization for the "*release order"* and the "*mark order as waiting*" activities). So the bars visualize time dependencies and causal dependencies between the several activities.

For extending the activity diagram to an object flow diagram, all conditions to object flows may be mapped by applying the above mentioned heuristic**: "**each condition having an involved thing type is a candidate for an object flow". In our example, the conditions*: payment is not authorized, payment is authorized*, *order is rejected*, *order is released*, *order is waiting*, *all articles in stock,* etc. are candidates for object flows. Finally, we can look at the actors of the operation types in the co-operation type schema and map them (in our case: *bookkeeping department*, *order department* and *stock clerk)* or an abstraction of these actors to "responsibilities" within swim lanes.

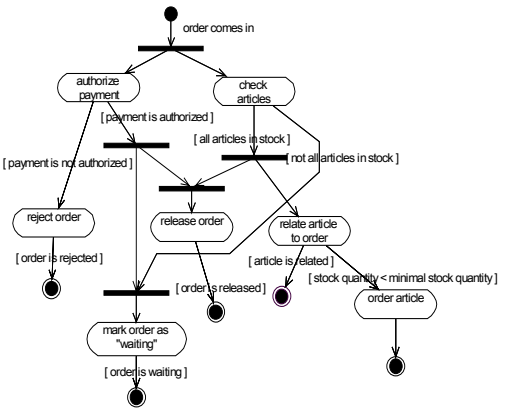

Figure 3: example transformed to an activity diagram

5. CONCLUSION AND FUTURE WORK

In this paper we have shown, that using the conceptual predesign model KCPM as an inter-lingua between natural language and conceptual modeling languages allows for a straightforward mapping of requirements specifications. In particular, different target model representations could be derived from the inter-lingua representation, e.g. there are similar sets of heuristics for mapping a KCPM co-operation type schema to Use Cases, state charts or Petri Net based models. This is due to the fact that all basic principles of dynamic modeling are covered by the KCPM Dynamic model. Clearly, the mapping process can and should be automated, a task that is actually in prog-

[3] Rational rose is a trade mark of Rational Cooperation. We have used Rational Rose Enterprise Edition 2001.

ress at our research lab: Up to now there is a complete automation of the mapping of requirements concerning static UoD properties (given in a somewhat normalized form of German) to an UML object model. Again KCPM acts as an inter-lingua of that process, so that also other kinds of structure models, e.g. all kinds of (extended) entity relationship models, may be derived.

Experiences gained from real world projects that we carried out in different application areas confirmed our expectation that the KCPM approach is well understood and accepted by end-users. Instead of a graphical representation of the KCPM schema, however, we used for that projects a glossary like notation which seems to meet better the capabilities of end-users. In addition, glossaries support better the process of requirements collection and validation (e.g. w.r.t. completeness). In fact, using glossaries, proved to be a very good decision concerning the representation of all structural and functional aspects. However, since more technically oriented people prefer figures we also introduced the graphical notation.

REFERENCES

[1] G. Booch, J. Rumbaugh, I. Jacobson, *The unified modeling language user guide*, Addison Wesley Publ. Comp., 1999.

[2] J.F.M. Burg, *Linguistic Instruments in Requirements Engineering*, IOS Press, Amsterdam, 1997.

[3] S. Ceri (ed.), *Methodology and Tools for Database Design*, North Holland 1983.

[4] C. J. Fillmore, The Case for Case, in E. Bach, R. T. Harms (eds.) *Universals in Linguistic Theory*. Holt, Rinehart and Winston, 1968.

[5] G. Fliedl, Ch. Kop, W. Mayerthaler, H.C. Mayr, Ch. Winkler: Guidelines for NL-Based Requirements Specifications in NIBA, in M. Bouzeghoub et. al. (eds.) *5th Int. Conf. on Applications of Natural Language to Information Systems (NLDB2000)*, LNCS 1959, Springer Verlag, 2000, pp. 251–264.

[6] M. Fowler, K. Scott: *UML konzentriert*. Addison-Wesley Publ, Comp. 1997.

[7] R. Kaschek, C. Kohl, H.C. Mayr: Cooperations - An Abstraction Concept Suitable for Business Process Reengineering, in Györkös, J. et.al. (eds.) *Re-Technologies for Information Systems, Proc. ReTIS'95*, R. Oldenburg Verlag, Wien, 1995.

[8] G. Kösters, H.-W. Six, M. Winter, Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications, *Journal of Requirements Engineering*, *Vol. 6 No. 1,* 2001, Springer Verlag, pp 3–17.

[9] Ch. Kop, H.C. Mayr, Conceptual Predesign – Bridging the Gap between Requirements and Conceptual Design, in *Proc. 3rd Int. Conf. on Requirements Engineering* ICRE'98, Colorado Springs, April 6-10, 1998.

[10] J.C.S. Leite, G.D.S. Hadad, J.H. Doorn and G.N. Kaplan, A Scenario Construction Process, *Journal of Requirements Engineering*, *Vol. 5 No. 1*, 2000, Springer Verlag, , pp. 38–61.

[11] W. Mayerthaler, G. Fliedl, Ch. Winkler, *Lexikon der Natürlichkeitstheoretischen Syntax und Morphosyntax*, Stauffenburg Verlag, Brigitte Narr, Tübingen, 1998.

[12] H.C. Mayr, Ch. Kop, A User Centered Approach to Requirements Modeling, *Proc. Modellierung 2002*, Lecture Notes in Informatics LNI p-12, GI-Edition, 2002, pp. 75-86.

[13] C. Rolland, C. Ben Achour, Guiding the Construction of Textual Use Case Specifications, *Data & Knowledge Engineering Journal*, *Vol. 25 No 1-2,* 1998, North Holland Elsevier Science Publ., pp. 125–160.

[14] C. Rolland et al., A proposal for a Scenario Classification Framework, *Journal of Requirements Engineering, Vol. 3 No. 1,* 1998, Springer Verlag, pp. 23–47.

[15] J. Rumbaugh et. al., *Objektorientiertes Modellieren und Entwerfen*, Carl Hanser Verlag, München Wien, 1993.

[16] G. Schneider, J. P. Winters, *Applying Use Cases a practical guide*, Addison Wesley Publ. Comp., 1998.

[17] I.-Y. Song, Developing Sequence Diagrams in UML, in H.S. Kunii et al. (eds.) *Proc. 20th Int. Conf. on Conceptual Modeling ER2001*, LNCS 2224, Springer Verlag Heidelberg, November 2001, pp. 368 – 382.

[18] K. Weidenhaupt et al., Scenarios in System Development–Current Practice, *IEEE Software*, *Vol. 15 No. 2*, 1998, pp. 34–45.

[19] H. Zhu, L. Jin, Scenario Analysis in an Automated Tool for Requirements Engineering, *Journal of Requirements Engineering*, *Vol. 5 No. 1*, 2000, Springer Verlag, pp. 2 – 22.

APPENDIX

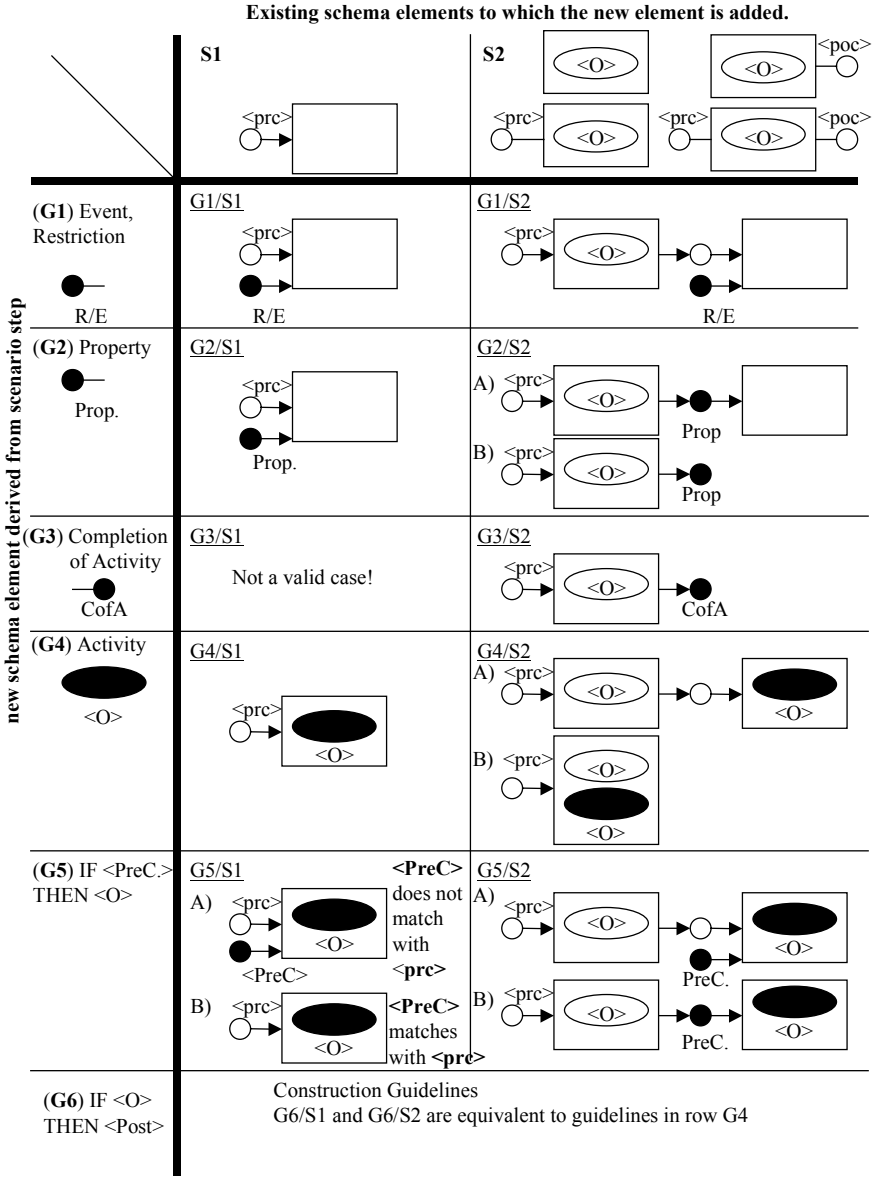

Figure 4: Construction guidelines for a valid cooperation type schema construction

Figure 4: cooperation type schema construction guidelines