Pattern Matching in Conceptual Models – A Formal Multi-Modelling Language Approach

Patrick Delfmann, Sebastian Herwig, Łukasz Lis, Armin Stein

University of Münster
European Research Center for Information Systems (ERCIS)
Leonardo-Campus 3
48149, Münster
Germany
{delfmann | herwig | lis | stein}@ercis.uni-muenster.de

Abstract: Recognizing patterns in conceptual models is useful for a number of purposes, for example revealing syntactical errors, model comparison, and identification of business process improvement potentials. In this contribution, we introduce a formal approach for the specification and matching of structural patterns in conceptual models. Unlike existing approaches, we do not focus on a certain application problem or a specific modelling language. Instead, our approach is generic making it applicable for any pattern matching purpose and any conceptual modelling language. In order to build sets representing structural model patterns, we define formal operations based on set theory, which can be applied to arbitrary models represented by sets. Besides a conceptual and formal specification of our approach, we present a prototypical modelling tool that shows its applicability through a particular application scenario.

1 Introduction

The structural analysis of conceptual models has multiple applications. To support modellers in their analyses, applying structural patterns to conceptual models is an established approach. Single conceptual models, for example, are analysed by use of typical error patterns in order to check for syntactical failures [Me07]. In the domain of Business Process Management (BPM), process models analysis helps identifying process improvement potentials [VTM08]. For example, applying structural model patterns to process models can help revealing changes of data medium during process execution (e.g., printing and retyping a document), redundant execution of process activities or application potentials of software systems. Whenever modelling is conducted in a distributed way, model integration is necessary to obtain a coherent view on the modelling domain. To find corresponding fragments and to evaluate integration opportunities, multiple models – generally of the same modelling language – can be compared with each other applying structural model pattern matching [GMS05]. Different model structures that typically represent equal real-world issues are identified and specified as structurally different, but semantically equal patterns. Counterparts of these patterns are searched via pattern matching in the models to be compared. If pattern counterparts are found in different models, these are marked as candidates for equivalent model sections. A subsequent comparison of their elements shows if their contents are equal as well. This way, structural pattern matching provides decision support in model comparison and integration. Model patterns have already been subject of research in the fields of database schema integration and workflow management, to give some examples. However, a literature review reveals that existing pattern matching approaches are limited to a specific domain or restricted to a single modelling language (cf. Section 2). We argue that the modelling community would benefit from a more generic approach, which is not restricted to particular modelling languages or application scenarios.

In this paper, we present a set theory-based model pattern matching approach, which is generic and thus not restricted regarding its application domain or modelling language. We base this approach on set theory as any model can be regarded as a set of objects and relationships – regardless of the modelling language or application domain. Set operations are used to construct any structural model pattern for any purpose. Therefore, we propose a collection of functions acting on sets of model elements and define set operators to combine the resulting sets of the functions (cf. Section 3). This way, we are able to specify structural model patterns for a given modelling language in form of expressions built of the proposed functions and operators. These pattern descriptions can be matched against conceptual models of this language resulting in sets of model elements, which represent particular pattern occurrences. As a specification basis, we use a generic meta-meta model being able to instantiate any modelling language. Consequently, a meta model-based specification of the modelling language the patterns are defined for is necessary for the application of our approach. In this paper, we provide an application example for Event-driven Process Chains (EPC) [Sc00] (cf. Section 4). Furthermore, we present a prototypical modelling tool implementation that shows the applicability of the approach. The example of Section 4 serves again as the application scenario (cf. Section 5). Finally, we conclude our paper and outline further need for research (cf. Section 6).

2 Related Work

Supporting the structural analysis of conceptual models, fundamental work is done in the field of graph theory addressing the problem of graph pattern matching [GMS05; Fu95; VVS06, VM97]. Based on a given graph, these approaches discuss the identification of structurally equivalent (homomorphism) or synonymous (isomorphism) parts of the given graph in other graphs. To identify such parts, several pattern matching algorithms are proposed, which make use of a pattern definition as comparison criteria to find corresponding parts in other graphs. The algorithms compute walks through the graphs in order to analyze its nodes and its structure. As a result, they identify patterns representing corresponding parts of the compared graphs. Thus, a pattern is based on a particular labelled graph section and is not predefined independently. Some approaches are limited to specific types of graphs (e.g., the approaches of [Fu95; VaVS06] are restricted to labelled directed graphs).

In the context of process models, so-called behavioural approaches have been proposed [Hi93; MAW08; Hi05]. Two process models are considered equivalent if they behave identically during simulation. This implies that the respective modelling languages possess formal execution semantics. Therefore, the authors focus on Petri Nets and other workflow modelling languages [DDM08]. Moreover, due to the requirement of model simulation, these approaches generally consider process models as a whole. Patterns as model subsets are only comparable if they are also executable. Hence, not every pattern – even if provided with formal execution semantics – can be used for matching.

In the domain of database engineering, various approaches have been presented, which address the problem of schema matching. Two input schemas (i.e., descriptions of database structures) are taken and mappings between semantically corresponding elements are produced [RB01]. These approaches operate on single elements only [LC00] or assume that the schemas have a tree-like structure [MBR01]. Recently, the methods developed in the context of database schema matching have been applied in the field of ontology matching as well [Au05]. Additionally, approaches explicitly dedicated to matching ontologies have been presented. They usually utilize additional context information (e.g., a corresponding collection of documents [SM01]), which is not given in standard conceptual modeling settings. Moreover, as schema-matching approaches operate on approximation-basis, similar structures – and not exact pattern occurrences – are addressed. Consequently, these approaches lack the opportunity of including explicit structure descriptions (e.g., paths of a given length or loops not containing given elements) in the patterns.

Design patterns are used in systems analysis and design to describe best-practice solutions for common recurring problems. Common design situations are identified, which can be modelled in various ways. The most desirable solution is identified as a pattern and recommended for further usage. The general idea originates from [AIS77], who identified and described patterns in the field of architecture. [Ga95] and [Fo02] popularized this idea in the domain of object-oriented systems design. Workflow patterns, that is patterns applied to workflow models, is another dynamically developing research domain regarding patterns [Aa03]. However, the authors do not consider pattern matching. Instead, the modeller is expected to adopt the patterns as best-practice and to apply them intuitively whenever a common problem situation is met. A methodical pattern matching support is not addressed.

Patterns are also proposed as an indicator for possible conflicts typically occurring in the modelling and model integration process. [Ha94] proposes a set of general patterns for Entity-Relationship Models (ERMs [Ch76]). On the one hand, these patterns depict possible structural errors that may occur. For such error patterns corresponding patterns are proposed, which provide correct structures. On the other hand, sets of model patterns are discussed, which possibly lead to conflicts while integrating such models into a total model. Similar work in the field of process modelling is done by [Me07]. Based on the analysis of EPCs, he detects a set of general patterns, which depict syntactical errors in EPCs. However, these two approaches focus on particular structural patterns for specific modelling languages rather than a pattern definition and matching approach for arbitrary modelling languages.

3 Specification of Structural Model Patterns

3.1 Sets as a Basis for Pattern Matching

The idea of our approach is to regard a conceptual model as a set of model elements. Here, we further distinguish between objects representing nodes and relationships representing edges interrelating objects. Starting from this set, pattern matches are searched by performing set operations on this basic set. By combining different set operations, the pattern is built up successively. Given a pattern definition, the matching process returns a set of model subsets representing the pattern matches found. Every match found is put into an own subset. The following example illustrates the general idea.

A pattern definition consists of three objects of different types that are interrelated with each other by relationships. A pattern match within a model is represented as a set containing three different objects and three relationships that connect them. To distinguish multiple pattern matches, each match is represented as a separate subset. Thus, the result of a pattern matching process is represented by a set of pattern matches (i.e., a set of sets, cf. Fig. 1).

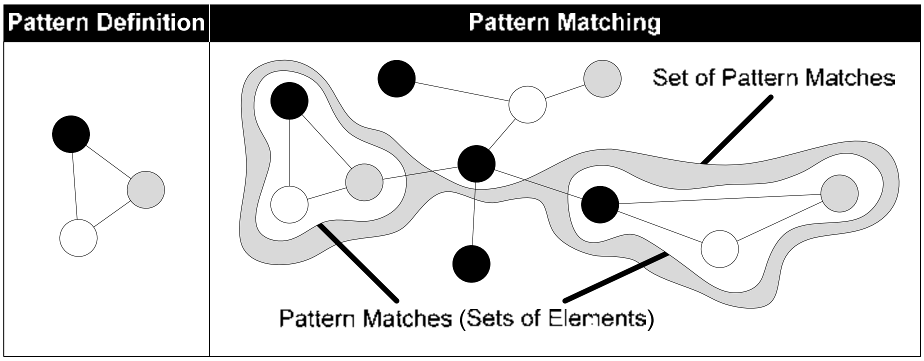

Fig. 1. Representation of Pattern Matches through Sets of Elements

3.2 Definition of Basic Sets

Therefore, as a basis for the specification of structural model patterns, we use a generic specification environment for conceptual modelling languages and models (cf. Fig. 2) applying the Entity-Relationship notation with (min,max)-cardinalities [ISO82]. Modelling languages typically consist of modelling objects that are interrelated through relationships (e.g., nodes and edges). In some modelling languages, relationships can be interrelated in turn (e.g., association classes in UML Class Diagrams [OMG09]).

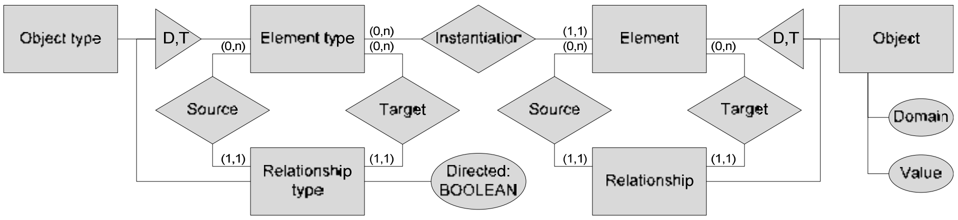

Fig. 2. Generic Specification Environment for Conceptual Modelling Languages and Models

Hence, modelling languages consist of *element types*, which are specialized into *object types* (i.e., nodes) and *relationship types* (e.g., edges and links). In order to allow relationships between relationships, the relationship type is defined as a specialization of the element type. Each relationship type has a *source* element type, from which it originates, and a *target* element type, to which it leads. Relationship types are either directed or undirected. Whenever the attribute *directed* is *FALSE*, the direction of the relationship type is ignored. The *instantiation* of modelling languages leads to models, which consist of particular *elements*. These are instantiated from their distinct element type. Elements are specialized into *objects* and *relationships*. Each of the latter leads from a *source element* to a *target element*. Objects can have *values* which are part of a distinct *domain*. For example, the value of an object "name" contains the string of the name (e.g., "product"). As a consequence, the domain of the object "name" has to be "string" in this case. Thus, attributes are considered as objects.

For the specification of structural model patterns we define the following sets, elements, and properties originating from the specification environment:

- E: set of all elements available; $e \in E$ is a particular element.
- $\boldsymbol{P}(E)$: power set of E.
- O: set of all objects available; $O \subseteq E$; $o \in O$ is a particular object.
- R: set of all relationships available; $R \subseteq E$; $r \in R$ is a particular relationship.
- A: set of all element types available; $a \in A$ is a particular element type.
- B: set of all object types available; $B \subseteq A$; $b \in B$ is a particular object type.
- C: set of all relationship types available; $C \subseteq A$; $c \in C$ is a particular relationship type.
- I: set of all instantiations available; $I \subseteq A \times E$; $(a,e) \in I$ is a particular instantiation.
- T: set of all relationship targets available; $T \subseteq E \times R$; $(e,r) \in T$ is a particular target.
- S: set of all relationship sources available; $S \subseteq E \times R$; $(e,r) \in S$ is a particular source.
- X: set of elements with $x \in X \subseteq E$.
- X_k: sets of elements with $X_k \subseteq E$ and $k \in \boldsymbol{N_0}$
- x_l: distinct elements with $x_l \in E$ and $l \in \boldsymbol{N_0}$
- Y: set of objects with $y \in Y \subseteq O$.
- Z: set of relationships with $z \in Z \subseteq R$.
- *directed(c)*: property *directed* of a particular relationship type c.
- *domain(o)*: property *domain* of a particular object o.
- *value(o)*: property *value* of a particular object o.
- n_x: positive natural number $n_x \in \boldsymbol{N_1}$

- R_d: set of all directed relationships available;
  $R_d \subseteq R$, $((c_d,r_d) \in I \wedge directed(c_d)=TRUE \wedge c_d \in C)\ \ \forall r_d \in R_d$
- T_d: set of all directed relationship targets available; $T_d \subseteq T$, $(r_d \in R_d)\ \ \forall (e,r_d) \in T$
- S_d: set of all directed relationship sources available; $S_d \subseteq S$, $(r_d \in R_d)\ \ \forall (e,r_d) \in S$
- T_u and S_u are undirected counterparts; $T_u = T \setminus T_d$ and $S_u = S \setminus S_d$

3.3 Definition of Set-modifying Functions

Building up structural model patterns successively requires performing set operations on these basic sets. In the following, we introduce predefined functions on these sets in order to provide a convenient specification environment for structural model patterns dedicated to conceptual models. Each function has a defined number of input sets and returns a resulting set. For every function, we specify the input and output sets and provide a formal specification based on predicate logic. In addition, we provide textual explanations where necessary. First, since a goal of the approach is to specify any structural pattern, we must be able to reveal specific properties of model elements (e.g., type, value, or value domain):

- $ElementsOfType(X,a) \subseteq E$ is provided with a set of elements X and a distinct element type a. It returns a set containing all elements of X that belong to the given type:

  $ElementsOfType(X,a)=\{x \in X | (a,x) \in I\}$
- $ObjectsWithValue(Y,valueY) \subseteq O$ takes a set of objects Y and a distinct value $valueY$. It returns a set containing all objects of Y whose values equal the given one:

  $ObjectsWithValue(Y,valueY)=\{y \in Y | value(y)=valueY\}$
- $ObjectsWithDomain(Y,domainY)) \subseteq O$ takes a set of objects Y and a distinct domain $domainY$. It returns a set with all objects of Y whose domains equal the given one:

  $ObjectsWithDomain(Y,domainY))=\{y \in Y | domain(y)=domainY\}$

Second, relations between elements have to be revealed in order to assemble complex pattern structures successively. Functions are required that combine elements and their relationships and elements that are related respectively.

- $ElementsWithRelations(X,Z) \subseteq \boldsymbol{P}(E)$ is provided with a set of elements X and a set of relationships Z. It returns a set of sets containing all elements of X and all relationships of Z, which are connected. Each occurrence is represented by an inner set:

  $EWR(x_1,Z)=\{z \in Z | (x_1,z) \in T \vee (x_1,z) \in S\} \cup \{x_1\},\ x_1 \in E$
  $ElementsWithRelations(X,Z)=\{EWR(x,Z)\},\ x \in X$
- $ElementsWithOutRelations(X,Z_d) \subseteq \boldsymbol{P}(E)$ is provided with a set of elements X and a set of relationships Z. It returns a set of sets containing all elements of X that are connected to outgoing relationships of Z, including these relationships. Each occurrence is represented by an inner set:

  $EWOR(x_1,Z_d)=\{z_d \in Z_d | (x_1,z_d) \in S_d\} \cup \{x_1\},\ x_1 \in E$
  $ElementsWithOutRelations(X,Z_d)=\{EWOR(x,Z_d)\},\ x \in X$

- *ElementsWithInRelations(X,Z)*$\subseteq$***P****(E)* is defined analogously:

  $EWIR(x_1,Z)=\{z_d \in Z_d | (x_1,z_d) \in T_d\} \cup \{x_1\},\ x_1 \in E$
  $ElementsWithInRelations(X,Z_d)=\{EWIR(x,Z_d)\},\ x \in X$

- *ElementsDirectlyRelatedInclRelations(X$_1$,X$_2$)*$\subseteq$***P****(E)* is provided with two sets of elements X_1 and X_2. It returns a set of sets containing all elements of X_1 and X_2 that are connected directly via relationships of *R*, including these relationships. The directions of the relationships given by their "Source" or "Target" assignment are ignored. Furthermore, the attribute "directed" of the according relationship types has to be *FALSE*. Each occurrence is represented by an inner set:

  $EDRIR(x_1,X_2)=\{x_2 \in X_2, z \in R_u | (x_1,z) \in S_u \wedge (x_2,z) \in T_u \vee (x_2,z) \in S_u \wedge (x_1,z) \in T_u\} \cup \{x_1\}$
  $ElementsDirectlyRelatedInclRelations(X_1,X_2)=\{EDRIR(x_1,X_2)\},\ x_1 \in X_1$

- *DirectSuccessorsInclRelations(X$_1$,X$_2$)*$\subseteq$***P****(E)* is provided with two sets of elements X_1 and X_2. It returns a set of sets containing all elements of X_1 and X_2 that are connected directly via relationships of *R*, including these relationships. The directions of the relationships are predefined, this is only relationships from elements of X_1 to elements of X_2 are considered. Each occurrence is represented by an inner set:

  $DSIR(x_1,X_2)=\{x_2 \in X_2, z \in R_d | (x_2,z) \in S_d \wedge (x_1,z) \in T_d\} \cup \{x_1\}$
  $DirectSuccessorsInclRelations(X_1,X_2)=\{DSIR(x_1,X_2)\},\ x_1 \in X_1$

Third, in order to construct model patterns representing recursive structures (e.g., a path of an arbitrary length consisting of alternating elements and relationships) the following functions are defined. For the specification of recursive structures, we make use of mathematical sequences that have to be transformed into sets. Therefore, we define an auxiliary function $Set((x_i))=\{x_i \in E | x_i \in (x_i)\} \subseteq E$.

- *Paths(X$_1$,X$_n$)*$\subseteq$***P****(E)* takes two sets of elements as input and returns a set of sets containing all sequences, which lead from any element of X_1 to any element of X_n. The directions of the relationships, which are part of the paths, are ignored. Furthermore, the attribute "directed" of the according relationship types has to be *FALSE*. The elements being part of the paths do not necessarily have to be elements of X_1 or X_n, but can also be of $E \backslash X_1 \backslash X_n$. Each path found is represented by an inner set:

  $PX(x_1,x_n)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \wedge ((x_i,x_{i+1}) \in S_u \vee (x_i,x_{i+1}) \in T_u))\ \forall 1 \leq i < n\}$
  $Paths(X_1,X_n)= \bigcup_{x_1 \in X_1, x_n \in X_n} PX(x_1,x_n)$

- *DirectedPaths(X$_1$,X$_n$)*$\subseteq$***P****(E)* is the directed counterpart of Paths. It returns only paths containing directed relationships of the same direction. Each such path found is represented by an inner set:

  $DPX(x_1,x_n)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \wedge (((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d\ \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
  $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d\ \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
  $\vee ((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d \wedge (x_{n-1},x_n) \in S_d\ \forall 1 \leq i \leq \lfloor n/2 \rfloor - 1)$
  $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d \wedge (x_n,x_{n-1}) \in T_d\ \forall 1 \leq i \leq \lfloor n/2 \rfloor - 1))\ \forall 1 \leq i < n\}$
  $DirectedPaths(X_1,X_n)= \bigcup_{x_1 \in X_1, x_n \in X_n} DPX(x_1,x_n)$

- *Loops(X)*$\subseteq$***P****(E)* takes a set of elements as input and returns a set of sets containing all sequences, which lead from any element of *X* to itself. The direction of relations and path elements are handled analogously to *Paths*. Each loop found is represented by an inner set:

$Loops(X)= \bigcup_{x\in X} PX(x,x)$

- $DirectedLoops(X)\subseteq \mathbf{P}(E)$ is defined analogously:

  $DirectedLoops(X)= \bigcup_{x\in X} DPX(x,x)$

In order to provide a convenient specification environment for structural model patterns, we define some additional functions that are derived from those already introduced:

- $ElementsWithRelationsOfType(X,Z_d,c_d)\subseteq \mathbf{P}(E)$ is provided with a set of elements X, a set of relationships Z_d and a distinct relationship type c_d. It returns a set of sets containing all elements of X and relationships of Z_d of the type c_d, which are connected. Each occurrence is represented by an inner set:

  $ElementsWithRelationsOfType(X,Z_d,c_d)=$
  $ElementsWithRelations(X,ElementsOfType(Z_d,c_d))$

- $ElementsWithOutRelationsOfType(X,Z_d,c_d)\subseteq \mathbf{P}(E)$ is provided with a set of elements X, a set of relationships Z_d and a distinct relationship type c_d. It returns a set of sets containing all elements of X that are connected to outgoing relationships of Z_d of the type c_d, including these relationships. Each occurrence is represented by an inner set:

  $ElementsWithOutRelationsOfType(X,Z_d,c_d)=$
  $ElementsWithOutRelations(X,ElementsOfType(Z_d,c_d))$

- $ElementsWithInRelationsOfType(X,Z_d,c_d)\subseteq \mathbf{P}(E)$ is defined analogously to *Elements-WithOutRelationsOfType*:

  $ElementsWithInRelationsOfType(X,Z_d,c_d)=$
  $ElementsWithInRelations(X,ElementsOfType(Z_d,c_d))$

- $ElementsWithNumberOfRelations(X,n_x)\subseteq \mathbf{P}(E)$ is provided with a set of elements X and a distinct number n_x. It returns a set of sets containing all elements of X, which are connected to the given number of relationships of R, including these relationships. Each occurrence is represented by an inner set:

  $EWNR(x)=\{r\in R \mid (x,r)\in T\vee(x,r)\in S\}\cup\{x\}$
  $ElementsWithNumberOfRelations(X,n_x)=\{EWNR(x) \mid |EWNR(x)|=n_x+1\}$

- $ElementsWithNumberOfOutRelations(X,n_x)\subseteq \mathbf{P}(E)$ and $ElementsWithNumberOfInRelations(X,n_x)\subseteq \mathbf{P}(E)$ are defined analogously:
  - $EWNIR(x)=\{r\in R_d \mid (x,r)\in T_d\}\cup\{x\}$
    $ElementsWithNumberOfInRelations(X,n_x)=\{EWNIR(x) \mid |EWNIR(x)|=n_x+1\}$
  - $EWNOR(x)=\{r\in R_d \mid (x,r)\in S_d\}\cup\{x\}$
    $ElementsWithNumberOfOutRelations(X,n_x)=\{EWNOR(x) \mid |EWNOR(x)|=n_x+1\}$

- $ElementsWithNumberOfRelationsOfType(X,c,n_x)\subseteq \mathbf{P}(E)$ is provided with a set of elements X, a distinct relationship type c and a distinct number n_x. It returns a set of sets containing all elements of X, which are connected to the given number of relationships of R of the type c, including these relationships. Each occurrence is represented by an inner set:

$EWNRT(x,c)=\{r \in R | (c,r) \in I \wedge ((x,r) \in T \vee (x,r) \in S)\} \cup \{x\}$
$ElementsWithNumberOfRelationsOfType(X,c,n_x)=$
$\{EWNRT(x,c) | \ |EWNRT(x,c)|=n_x+1\}$

- $ElementsWithNumberOfOutRelationsOfType(X,c_d,n_x) \subseteq \mathbf{P}(E)$ and $ElementsWithNumberOfInRelationsOfType(X,c_d,n_x) \subseteq \mathbf{P}(E)$ are defined analogously:
  - $EWNIRT(x,c_d)=\{r \in R_d | (c_d,r) \in I \wedge (x,r) \in T_d\} \cup \{x\}$
    $ElementsWithNumberOfInRelationsOfType(X,c_d,n_x)=$
    $\{EWNIRT(x,c_d) | \ |EWNIRT(x,c_d)|=n_x+1\}$
  - $EWNORT(x,c_d)=\{r \in R_d | (c_d,r) \in I \wedge (x,r) \in S_d\} \cup \{x\}$
    $ElementsWithNumberOfOutRelationsOfType(X,c_d,n_x)=$
    $\{EWNORT(x,c_d) | \ |EWNORT(x,c_d)|=n_x+1\}$
- $PathsContainingElements(X_1,X_n,X_c) \subseteq \mathbf{P}(E)$ is provided with three sets of elements $X_1,X_n,$ and X_c. It returns a set of sets containing elements that represent all paths from elements of X_1 to elements of X_n, which each contain at least one element of X_c. The direction of relations and path elements are handled analogously to *Paths*. Each path found is represented by an inner set:

$PCE(x_1,x_n,X_c)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \wedge \exists x_c \in \{x_2,\ldots,x_{n-1}\} \wedge$
$((x_i,x_{i+1}) \in S_u \vee (x_i,x_{i+1}) \in T_u) \ \forall 1 \leq i < n\}$
$PathsContainingElements(X_1,X_n,X_c)= \bigcup_{x_1 \in X_1, x_n \in X_n} PCE(x_1,x_n,X_c)$

- $DirectedPathsContainingElements(X_1,X_n,X_c) \subseteq \mathbf{P}(E)$, $PathsNotContainingElements(X_1,X_n,X_c) \subseteq \mathbf{P}(E)$, and $DirectedPathsNotContainingElements(X_1,X_n,X_c) \subseteq \mathbf{P}(E)$ are defined analogously:
  - $DPCE(x_1,x_n,X_c)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \wedge \exists x_c \in \{x_2,\ldots,x_{n-1}\} \wedge$
    $(((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d \ \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
    $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d \ \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
    $\vee ((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d \wedge (x_{n-1},x_n) \in S_d \ \forall 1 \leq i \leq \lfloor n/2 \rfloor -1)$
    $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d \wedge (x_n,x_{n-1}) \in T_d \ \forall 1 \leq i \leq \lfloor n/2 \rfloor -1)) \ \forall 1 \leq i < n\}$
    $DirectedPathsContainingElements(X_1,X_n,X_c)= \bigcup_{x_1 \in X_1, x_n \in X_n} DPCE(x_1,x_n,X_c)$
  - $PNCE(x_1,x_n,X_c)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \setminus X_c \wedge ((x_i,x_{i+1}) \in S_u \vee (x_i,x_{i+1}) \in T_u) \forall 1 \leq i < n\}$
    $PathsNotContainingElements(X_1,X_n,X_c)= \bigcup_{x_1 \in X_1, x_n \in X_n} PNCE(x_1,x_n,X_c)$
  - $DPNCE(x_1,x_n,X_c)=\{Set((x_1,x_2,\ldots,x_n)) | x_2,\ldots,x_{n-1} \in E \setminus X_c$
    $\wedge (((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
    $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d \forall 1 \leq i \leq \lfloor n/2 \rfloor)$
    $\vee ((x_{2i-1},x_{2i}) \in S_d \wedge (x_{2i+1},x_{2i}) \in T_d \wedge (x_{n-1},x_n) \in S_d \forall 1 \leq i \leq \lfloor n/2 \rfloor -1)$
    $\vee ((x_{2i},x_{2i-1}) \in T_d \wedge (x_{2i},x_{2i+1}) \in S_d \wedge (x_n,x_{n-1}) \in T_d \forall 1 \leq i \leq \lfloor n/2 \rfloor -1)) \forall 1 \leq i < n\}$
    $DirectedPathsNotContainingElements(X_1,X_n,X_c)= \bigcup_{x_1 \in X_1, x_n \in X_n} DPNCE(x_1,x_n,X_c)$
- $LoopsContainingElements(X,X_c) \subseteq \mathbf{P}(E)$, $DirectedLoopsContainingElements(X,X_c) \subseteq \mathbf{P}(E)$, $LoopsNotContainingElements(X,X_c) \subseteq \mathbf{P}(E)$, and $DirectedLoopsNotContainingElements(X,X_c) \subseteq \mathbf{P}(E)$ are defined analogously:
  - $LoopsContainingElements(X,X_c)= \bigcup_{x \in X} PCE(x,x,X_c)$

- *DirectedLoopsContainingElements*$(X,X_c) = \bigcup_{x\in X} DPCE(x,x,X_c)$
- *LoopsNotContainingElements*$(X,X_c) = \bigcup_{x\in X} PNCE(x,x,X_c)$
- *DirectedLoopsNotContainingElements*$(X,X_c) = \bigcup_{x\in X} DPNCE(x,x,X_c)$

3.4 Definition of Set Operators for Sets of Sets

By nesting the functions introduced above, it is possible to build up structural model patterns successively. The results of each function can be reused adopting them as an input for other functions. In order to combine different results, the basic set operators *union* (∪), *intersection* (∩), and *complement* (\) can be used generally. Since it should be possible to combine not only sets of pattern matches (i.e., sets of sets) but also the pattern matches themselves, this is the inner sets, we define additional set operators. These operate on the inner sets of two sets of sets respectively (cf. Table 1).

Basic Sets	Operator Definition	Operator Symbol
$F,G \subseteq \mathbf{P}(E)$, $f \in F$, $g \in G$	$Join(F,G)=\{f \cup g \mid \exists e \in E: e \in f \wedge e \in g\}$	$F \mathbin{\dot\cup} G$
$F,G \subseteq \mathbf{P}(E)$, $f \in F$, $g \in G$	$InnerIntersection(F,G)=\{f \cap g\}$	$F \mathbin{\dot\cap} G$
$F,G \subseteq \mathbf{P}(E)$, $f \in F$, $g \in G$	$InnerComplement(F,G)=\{f \setminus g \mid \exists e \in E: e \in f \wedge e \in g\}$	$F \mathbin{\boldsymbol{\setminus}} G$
$F \subseteq \mathbf{P}(E)$, $f \in F$	$SelfUnion(F)=\bigcup_{f\in F} f$	$\bigcup F$
$F \subseteq \mathbf{P}(E)$, $f \in F$	$SelfIntersection(F)=\bigcap_{f\in F} f$	$\bigcap F$

Table 1. Set Operators for Sets of Sets

The *Join* operator performs a *Union* operation on each inner set of the first set with each inner set of the second set. Since we regard patterns as cohesive, only inner sets that have at least one element in common are considered. The *InnerIntersection* operator *intersects* each inner set of the first set with each inner set of the second set. The *Inner-Complement* operator applies a *complement* operation to each inner set of the first outer set combined with each inner set of the second outer set. Only inner sets that have at least one element in common are considered.

As most of the functions introduced in Section 3.3 expect simple sets of elements as inputs, we introduce further operators that turn sets of sets into simple sets. The *Self-Union* operator merges all inner sets of one set of sets into a single set performing a *union* operation on all inner sets. The *SelfIntersection* operator performs an *intersection* operation on all inner sets of a set of sets successively. The result is a set containing elements that each occur in all inner sets of the original outer set.

4 Application of Structural Model Patterns

To illustrate the usage of the set functions, we apply our approach to an EPC application scenario. In the scenario, the approach is applied to complex syntax verification in EPCs.

Therefore, we regard a simplified modelling language of EPCs. Models of this language consist of the object types function, event, AND connector, OR connector, and XOR connector (i.e., B={function, event, AND, OR, XOR}). Furthermore, EPCs consist of different relationship types that lead from any object type to any other object type, except from function to function and from event to event. All these relationship types are directed, (i.e., *c.directed=TRUE* $\forall c \in C$).

A common error in EPCs is that decisions (i.e., XOR or OR splits) are modelled successively to an event. Since events are passive element types of an EPC, they are not able to make a decision [Sc00]. Hence, any directed path in an EPC that reaches from an event to a function and contains no further events or functions but an XOR or OR split is a syntax error. In order to reveal such errors, we specify the following exemplary structural model pattern:

DirectedPathsNotContainingElements (ElementsOfType (O, 'Event'), ElementsOfType (O, 'Function'), (ElementsOfType (O, 'Event') UNION ElementsOfType (O, 'Function')))	1
INTERSECTION	
DirectedPathsContainingElements (ElementsOfType (O, 'Event'), ElementsOfType (O, 'Function'),	2
((ElementsOfType (O, 'OR') UNION ElementsOfType(O, 'XOR'))	3
COMPLEMENT	
(O INNER_INTERSECTION (ElementsWithNumberOfOutRelations ((ElementsOfType (O, 'XOR') UNION ElementsOfType (O, 'OR')), 1) UNION ElementsWithNumberOfOutRelations ((ElementsOfType (O, 'XOR') UNION ElementsOfType (O, 'OR')), 0)))))	4

The first expression (cf. 1st block) determines all paths that start with an event and end with a function and do not contain any further functions or events. The result is intersected with all paths starting with an event and ending with a function (cf. 2nd block) that contain OR and/or XOR connectors (cf. 3rd block), but only those that are connected to 2 or more outgoing relationships. Thus, these XORs and ORs are subtracted by XORs and ORs that are only connected to one or less relationship(s) (cf. 4th block). Summarizing, all paths are returned that lead from an event to a function not containing any further events and functions, and that contain splitting XOR and/or OR connectors (cf. Section 5 for implementation issues and exemplary results). This way, any syntax error pattern can be specified and applied to any model base.

5 Tool Support

In order to show the feasibility of the approach, we have implemented a plug-in for a meta modelling tool that was available from a former research project. The tool consists of a meta modelling environment that is based on the generic specification approach for modelling languages shown in Fig. 2.

The plug-in provides a specification environment for structural model patterns, which is integrated into the meta modelling environment of the tool, since the patterns are dependent on the respective modelling language. All basic sets, functions, and set operators introduced in Section 3 are provided and can be used to build up structural model patterns successively. In order to gain a better overview over the patterns, they are displayed and edited in a tree structure. The tree-structure is built up through drag-and-drop of the basic sets, functions and set operators. Whenever special characteristics of an according modelling language (function, event etc.) or variables such as numeric values or names are used for the specification, this is expressed by using a “variable” element. The variable element, in turn, is instantiated by selecting a language-specific characteristic from a menu or by entering a particular value (such as “2”).

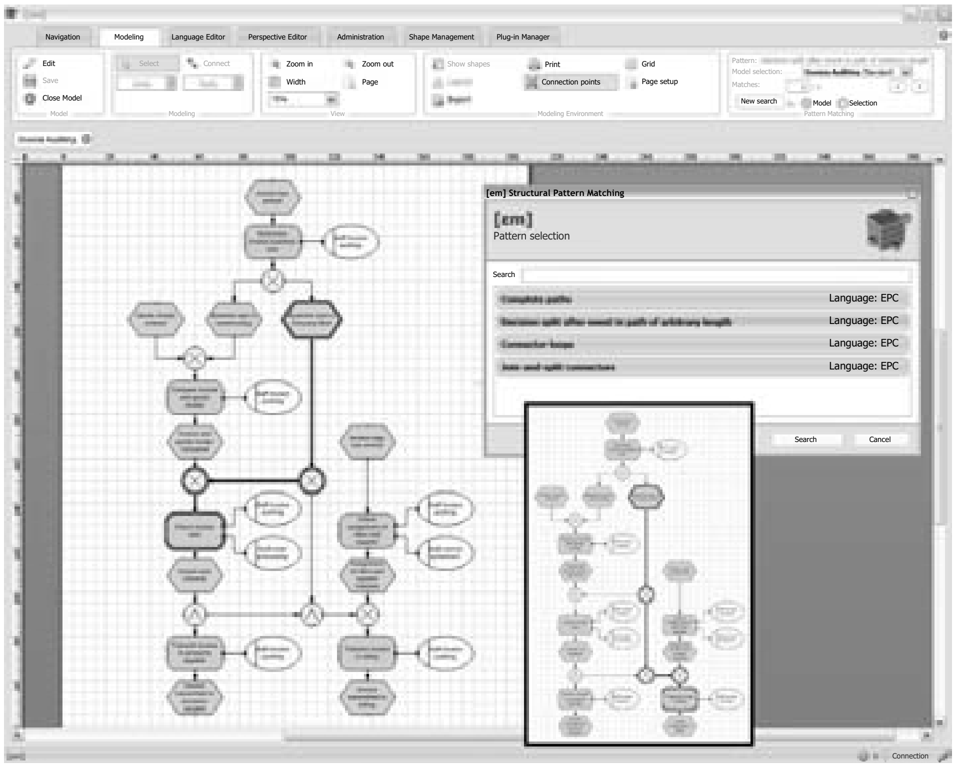

Fig. 3. Result of the Pattern Matching Process of “Decision split after event...”

The patterns specified can be applied to any model that is available within the model base and that was developed with the according modelling language. Fig. 3 shows an exemplary model that was developed with the modelling language of EPCs and that contains a syntax error consisting of a decision split following an event. The structural model pattern matching process is started by selecting the appropriate pattern to search for. Every match found is displayed by marking the according model section. The user can switch between different matches. In our example, two matches are found, as the decision split following the event leads to two different paths (the second match is shown in the lower right corner of Fig. 3).

6 Conclusion and Outlook

Supporting model analysis by a generic pattern matching approach is promising, as it is not restricted to a particular problem area or modelling language. A first rudimentary evaluation through implementation and exemplary application of the approach has shown its general feasibility. Nevertheless, there still remains need for further research.

In the short term, we will focus on completing the evaluation of the presented approach. Although our current prototypical implementation already shows its general feasibility, further evaluation of our approach is necessary. We will conduct a series of with-without experiments in real-world scenarios. They will show if the presented function set is complete, if the ease of use is satisfactory for users not involved in the development of the approach, and if the application of the approach actually leads to an improved model analysis support. Although we strongly believe that our tool-implemented approach will inevitably support modellers in the task of model analysis and integration, this needs to be objectively proven.

Medium-term research will address further applications for the structural model pattern matching approach presented here. For instance, we will question if modelling conventions on the basis of structural model patterns that are provided prior to modelling are able to increase the comparability of conceptual models.

References

[Au05] Aumueller, D., Do, H.-H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data, New York, 2005; pp. 906-908.

[Aa03] van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.; Barros, A. P.: Workflow Patterns. In: Distributed and Parallel Databases 14 (2003) 3; pp. 5-51.

[AIS77] Alexander, C.; Ishikawa, S.; Silverstein, M. A.: Pattern Language. New York, 1977.

[Ch76] Chen, P.P.-S.: The Entity-Relationship Model: Toward a Unified View of Data. In: ACM Transactions on Database Systems 1 (1976) 1; pp. 9-36.

[DDM08] van Dongen, B. F.; Dijkman, R.; Mendling, J.: Measuring similarity between business process models. In (Bellahsene, Z.; Léonard, M. eds.): Proceedings of the 20th International Conference on Advanced Information Systems Engineering, Montpellier, 2008; pp. 450-464.

[Fo02] Fowler, M.: Patterns of Enterprise Application Architecture. Reading, 2002.

[Fu95] Fu, J.: Pattern matching in directed graphs. In (Galil, Z.; Ukkonen, E. eds.): Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching, Espoo, 1995; pp. 64-77.

[Ga95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. New York, 1995.

[GMS05] Gori, M.; Maggini M.; Sarti, L.: The RW2 algorithm for exact graph matching. In (Singh, S.; Singh, M.; Apté, C.; Perner, P. eds.): Proceedings of the 4th International Conference on Advances in Pattern Recognition, Bath, 2005; pp. 81-88.

[Ha94] Hars, A.: Reference Data Models - Foundations of Efficient Data Modeling [In German: Referenzdatenmodelle. Grundlagen effizienter Datenmodellierung]. Wiesbaden, 1994.

[Hi05] Hidders, J.; Dumas, M.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Verelst, J.: When are two workflows the same? In (Atkinson, M.; Dehne, F. eds.): Proceedings of the 11th Australasian Symposium on Theory of Computing, Newcastle, 2005; pp. 3-11.

[Hi93] Hirschfeld, Y.: Petri nets and the equivalence problem. In (Börger, E.; Gurevich Y.; Meinke, K. eds.): Proceedings of the 7th Workshop on Computer Science Logic, Swansea, 1993; pp. 165-174.

[ISO82] ISO: Concepts and Terminology for the conceptual Schema and the Information Base. Technical report ISO/TC97/SC5/WG3, 1982.

[LC00] Li, W.; Clifton, C.: SemInt: a tool for identifying attribute correspondences in heterogeneous databases using neural network. In: Data & Knowledge Engineering 33 (2000) 1; pp. 49-84.

[MBR01] Madhavan, J.; Bernstein, P. A.; Rahm, E.: Generic schema matching with Cupid. In (Apers, P. M. G.; Atzeni, P.; Ceri, S.; Paraboschi, S.; Ramamohanarao, K.; Snodgrass, R. T. eds.): Proceedings of the 27th International Conference on Very Large Data Bases, Rome, 2001; pp. 49-58.

[MAW08] de Medeiros, A. K. A.; van der Aalst, W. M. P.; Weijters, A. J. M. M.: Quantifying process equivalence based on observed behavior. In: Data & Knowledge Engineering 64 (2008) 1; pp. 55-74.

[Me07] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Doctoral Thesis, Vienna University of Economics and Business Administration. Vienna, 2007.

[OMG09] Object Management Group (OMG): Unified Modeling Language (OMG UML), Infrastructure, V2.1.2, http://www.omg.org/docs/formal/07-11-04.pdf, 2009.

[RB01] Rahm, E.; Bernstein, P. A.: A Survey of approaches to automatic schema matching. In: The VLDB Journal – The International Journal on Very Large Data Bases 10 (2001) 4; pp. 334-350.

[Sc00] Scheer, A.-W.: ARIS – Business Process Modelling. 3rd Edition. Berlin et al., 2000.

[SM01] Stumme, G., Mädche, A.: FCA-Merge: Bottom-up merging of ontologies. In (Nebel, B. ed.): Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, 2001; pp. 225-230.

[VM97] Valiente, G.; Martínez, C.: An Algorithm for Graph Pattern-Matching. In (Baeza-Yates, R.; Ziviani, N. eds.): Proceedings of the 4nd South American Workshop on String Processing, Brighton, 2006; pp. 180-197.

[VVS06] Varró, G.; Varró, D.; Schürr, A.: Incremental Graph Pattern Matching - Data Structure and Initial Experiments. In (Margaria, T.; Padberg, J.; Taentzer, G. eds.): Proceedings of the 2nd International Workshop on Graph and Model Transformation, Brighton, 2006.

[VTM08] Vergidis, K.; Tiwari, A.; Majeed, B.: Business process analysis and optimization: beyond reengineering. In: IEEE Transactions on Systems, Man, and Cybernetics 38 (2008) 1; pp. 69-82.