SEMANTIC ANNOTATION OF EPC MODELS IN ENGINEERING DOMAINS BY EMPLOYING SEMANTIC PATTERNS*

Andreas Bögl, Michael Schrefl
Department of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria

Gustav Pomberger
Department of Business Informatics – Software Engineering
Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria

Norbert Weber
Siemens AG, Corporate Technology-SE 3, Otto-Hahn-Ring 6, Munich, Germany

Keywords: Semantic Annotation, Semantic Patterns, Semantic EPC Models, Process Ontology.

Abstract: Extracting process patterns from EPC (Event-Driven Process Chain) models requires to perform a semantic analysis of EPC functions and events. An automated semantic analysis faces the problem that an essential part of the EPC semantics is bound to natural language expressions in functions and events with undefined process semantics. The semantic annotation of natural language expressions provides an adequate approach to tackle this problem. This paper introduces a novel approach that enables an automated semantic annotation of EPC functions and events. It employs semantic patterns to analyze the textual structure of natural language expressions and to relate them to instances of a reference ontology. Thus, semantically annotated EPC model elements are input for subsequent semantic analysis.

1 INTRODUCTION

Designing processes is a sophisticated and cognitive task for process designers, usually supported by tools, such as the ARIS Toolset. Process designs are represented as process models in a particular language, usually depicted by a graph of activities and their casual dependencies. The Event-driven Process Chain (EPC) modeling language (Keller et al., 1992) has gained a broad acceptance and popularity both in research and in practice.

Engineering processes are characterized by an identifiable progression from requirements, through specification to design and implementation. Each of these phases comprises context specific tasks, conducted similarly in different engineering domains (Moore, 2000), e.g. the task *"Design Architecture"* usually succeeds to task *"Identify Requirements"*.

Similarly conducted tasks in different engineering domains motivate the extraction of process patterns, tracing back to obviously existing structural analogies in process models describing various domains in engineering domains (Bögl et al., 2008).

A process pattern represents a common or best practice solution to solve a particular problem in a certain context. Hence, it might assist process modelers for constructing high quality process solutions. Extracting process patterns requires to compare process models either human-driven or automatically. Prerequisites for an automatable comparison of conceptual models are discussed in Pfeiffer (2005). Most emphasized prerequisites refer to a formal semantics of modeling language and constructs. Additionally, a reference ontology should capture the real world semantics.

EPC models express the model element structure in terms of the meta language constructs which are

*The research presented in this paper was funded by Siemens AG, Corporate Technology – SE 3, Munich.

Bögl A., Schrefl M., Pomberger G. and Weber N. (2008).
SEMANTIC ANNOTATION OF EPC MODELS IN ENGINEERING DOMAINS BY EMPLOYING SEMANTIC PATTERNS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - AIDSS*, pages 106-115
DOI: 10.5220/0001716601060115

Functions (F), Events(E) and *Connectors* (and, xor, or). Natural language expressions describe the inner process semantics of functions and events. Depending on the sequence of words and on its semantics, different meanings may arise. For example, *"Define Software Requirements with Customer"* has a different meaning than *"Define Software Requirements for Customer"*. The former expression means that software requirements are defined in cooperate manner with a customer; the latter one indicates the customer as output target of the performed activity.

Semantic annotation of EPC models constitutes an appropriate solution to tackle the problem of different meanings but raises the problems to determine the process semantics of lexical terms used in natural language expressions and to link them to reference ontology instances. The process semantics of lexical terms represents process tasks executed on process objects, for instance.

Linking lexical terms to reference ontology instances requires the availability of corresponding reference ontology instances. In case of non-corresponding instances, semantic annotation also entails to populate a reference ontology with new instances derived from lexical terms.

Natural language expressions follow naming conventions or standards that represent guidelines for naming EPC functions/events (Schütte, 1998). If a naming convention is used, it is clear what a lexical term expresses.

For example, a typical suggested naming convention for an EPC function is the template *Task* followed by a *Process Object* that specifies the semantics of natural language expressions such as *"Define Project Plan"*; *"Define"* has the semantics of a task, the lexical term *"Project Plan"* indicates the semantics of a process object. We analyzed about 5,000 EPC functions/events in engineering domains. We experienced that the suggested recommendations do not fully cover the semantics of used natural language expressions. These investigations addionally triggers the neccessity of a formal notation that enables to express or specify a guideline. Formalized guidelines enable verifying natural language expressions against a set of predefined conventions.

Based on our investigations for clarifying the semantics of natural language expressions used for naming EPC functions/events, we introduce a set of guidelines that extend traditional naming conventions. The introduced guidelines are expressed through semantic pattern descriptions. Semantic pattern descriptions represent semantic templates that bridge the gap between informal and formal representation. Formal representation refers to concepts specified by a reference ontology. Semantic pattern descriptions are either defined for EPC functions or for EPC events. A semantic pattern description has a pattern template, that is represented in the form *Function(Context)[Task; Process Object; Parameter]* for an EPC function.

Context is the name of an engineering domain the analyzed EPC function is associated with, the concept *Task* represents the activity being performed on an instance of the concept *Process Object*, the concept *Parameter* captures an instance of a *Process Object* optionally for executing an EPC function. To represent the semantics of a function consists in instantiating a semantic pattern template by binding a lexical term to the variables *Context*, *Task*, *Process Object* and *Parameter*. For example, the instantiated semantic pattern template *Function(Software) [Task: "Define"; Process Object: "Quality Goal"]* defines the process semantics of the EPC function *"Define Quality Goal" in* the context *"Software"*.

Same or similar knowledge in EPC functions/events can be expressed in alternative ways due to the freedom of modeling (e.g. usage of synonyms, abbreviates etc.). Consequently, alternative natural language expressions may refer to same process semantics. For example, the EPC functions *"Define SW Requirements"* and *"Define Requirements for Software"* refer to the same process semantics.

For that reason each semantic pattern description defines a set of lexical structures and analysis rules. A lexical structure is defined by a sequence of word classes (e.g. Verb, Noun, Preposition). The lexical structure LS:= [$\text{Verb}_{\text{Task}}$] [$\text{NounGroup}_{\text{ProcessObject}}$] captures a natural language expression such as *Define* [$\text{Verb}_{\text{Task}}$] *Quality Goal* [$\text{NounGroup}_{\text{ProcessObject}}$]. A lexical structure instantiates a semantic pattern template by applying analysis rules. They define how to map lexical structures onto the reference ontology concepts specified in a semantic pattern template.

An instance of a reference ontology concept has an unique identifier which may have several textual counterparts. For example, a process object with the OID=123 has the textual counterparts *{"Software Requirements", "SW Requirements", "Requirements for Software"}*.

To separate meaning from its lexical representation the reference ontology is split into two layers. Layer one represents the lexical knowledge base that captures commonsense vocabulary used in natural language expressions;

layer two represents the semantic domain for those vocabulary by providing process ontology concepts and relations defined in the process knowledge base.

The paper is organized as follows: The upcoming section introduces the reference ontology, Section 3 deals with the main contribution of this paper, the semantic annotation process. In Section 4, we discuss related approaches. Finally, a conclusion is given in Section 5.

2 REFERENCE ONTOLOGY

The reference ontology provides concepts and relations whose instances capture both the vocabulary (lexical knowledge) used to describe EPC functions/events and its process semantics (process knowledge). Thus it serves as knowledge base for the semantic annotation of EPC functions/events. Lexical knowledge comprises morphological and syntactic knowledge about used vocabulary. Morphological knowledge relates to word inflections such as single/plural form of a noun, past tense of verbs. Syntactic knowledge comprises word class information and binary features such as countable/uncountable. Additionally, the lexical knowledge base explicitly considers context information and the relationships *isAcronymOf* and *isSynonymTo* between entries. Context information refers to engineering domains such as software or hardware development.

Process knowledge represents the semantic domain for natural language expressions. The process semantics is expressed by instances of process ontology concepts such as *task*, *process object* and its semantic relationships such as *isPartOf* or *isSubclassOf*. Figure 1 sketches the overall architecture. Natural language expressions are used for naming EPC functions/events. The process semantics implicitly described in an EPC function/event is captured in the process knowledge base, its textual representation is captured in the lexical knowledge base.

2.1 Lexical Knowledge Base

The rationale behind the lexical knowledge base is to provide a lightweight, controlled domain vocabulary for semantically annotated EPC functions/events.

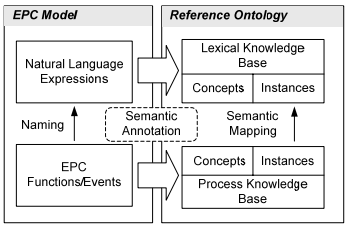

Figure 1: Coherence between Reference Ontology and EPC Functions/Events.

Publicly available resources such as WordNet (W3C, 2006) may provide commonsense vocabulary, but cannot be considered fully suitable for capturing domain specific vocabulary. In general, such resources are open world dictionaries, comprising several hundred thousand open world entities and semantic relationships. It is an unreasonable demand for a process designer to maintain this vocabulary. A domain specific controlled vocabulary within an engineering domain usually comprises several hundred entities that can be maintained easier. Nevertheless, WordNet plays a vital role for the semantic annotation of EPC functions/events. Its purpose is further discussed in section 3 of this paper.

The conceptual design of the lexical knowledge base relies on the analysis of natural expressions used for naming EPC functions/events. Figure 2 illustrates the structure of a natural language expression.

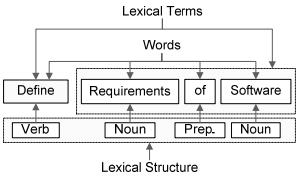

Figure 2: Structure of Natural Language Expressions.

A natural language expression used for naming is a composition of words each belonging to a word class. The sequence of word classes specifies the lexical structure of the natural language expression. In general, a lexical term represents a cluster of words belonging to equal word classes. For example, the word *"Define"* belongs to the word class verb, since there is only one occurrence of this word class, the lexical term only consists of the word *"Define"*. Besides this general rule, an *"and"* conjunction used in a natural language expression requires special attention. An "and" conjunction results in a separation of lexical terms. For example, let us replace the preposition "of" with an "and" conjunction in Figure 2. In this case, the natural language expression would comprise the two

additional lexical terms *"Requirements"* and *"Software"*.

The lexical knowledge base consists of lexical entries. A lexical entry is either single or multi-structured. A single structured lexical entry is represented by exactly one word; a multi-structured lexical entry consists of several words. One may recognize the analogy to a lexical term. Hence, a lexical entry represents a lexical term captured in the lexical knowledge base.

Let us consider Figure 3 that provides an example for the multi-structured lexical entry *"Software Requirements"* and for the lexical mapping between lexical and process knowledge base. The right section illustrates a part of a process ontology that captures the semantics of the lexical entry *"Software Requirements"* realized by a lexical mapping between a lexical entry and a process ontology instance. The meaning of this entry can be interpreted as follows: *"Software Requirements"* is a process object and it is a specialization of the process object *"Requirements" (<OID 3>).*

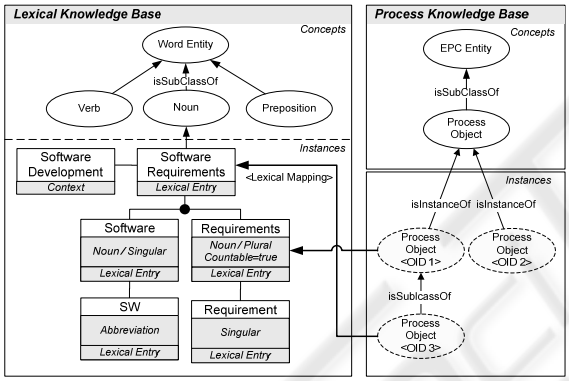

Figure 3: Example for Multi-Structured Lexical Entry.

2.2 Process Knowledge Base

The concepts and relations of process knowledge result from the analysis of natural language terms describing EPC functions/events and its associated meanings. Figure 4 depicts the process ontology concepts for capturing the semantics of EPC functions/Events.

The top level concept EPC Entity classifies a lexical term either into a Task(T), or a Process Object(PO) or a State(S) concept. A Process Object represents a real or an abstract thing being of interest within a (business process) domain. According to Rosemann (1995, 177 et seq), the concept for describing a Process Object has the semantic relations isPartOf (e.g. Project Handbook isPartOf Development Project), isSublcassOf (e.g. Software Project isSubClassOf Project) and migratesTo (e.g. Software Requirements *migratesTo* Software Architecture). A Task can be performed manually by a human or electronically by a service (e.g. Web Service) for achieving a desired (business) objective. It can be specified at different levels of abstraction, refinements or specializations that are expressed by the semantic relationship hasSubTask.

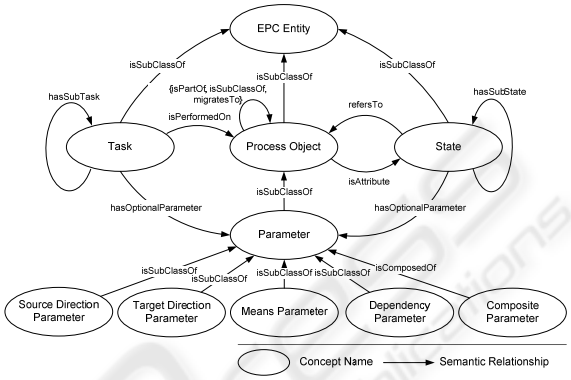

Figure 4: Process Ontology for Capturing the Semantics of EPC Functions/Events.

A State always refers to a Process Object indicating the state that results from performing a Task on a Process Object. Parameters are process objects that are relevant for a task execution or a state description. The process ontology comprises the four parameter types Source Direction Parameter, Target Direction Parameter, Means Parameter and Dependency Parameter.

A Source Direction Parameter defines a source process object, e.g. *"Derive Quality Goal* from *Specification Document"* indicates *"Specification Document"* as a Source Direction Parameter. A Target Direction Parameter denotes a recipient process object within a function execution, e.g. the function *"Rework Specification* for *Project Plan"* specifies *"Project Plan"* as a Target Direction Parameter. A Means Parameter semantically describes a process object as an input requirement for task execution. For example, *"Rework Specification* with *Software Goals"* indicates *"Software Goals"* as a Means Parameter. A Dependency Parameter indicates that executing a task on a process object depends on an additional process object, e.g. the function *"Decide Quality Measure Upon Review Status"* specifies *"Review Status"* as a Dependency Parameter.

Additionally, a function/event may contain a composition of parameters, expressed by the concept CompositeParameter. For example, *"Rework Specification with Software Goals for Project Handbook"* is a composition of the two parameters

"Software Goals" (Means Parameter) and *"Project Handbook"* (Target Direction Parameter).

3 SEMANTIC ANNOTATION PROCESS

Input are natural language expressions used for naming EPC functions/events. The semantic annotation process comprises the four stages as depicted in Figure 5 that are discussed in the following subsections.

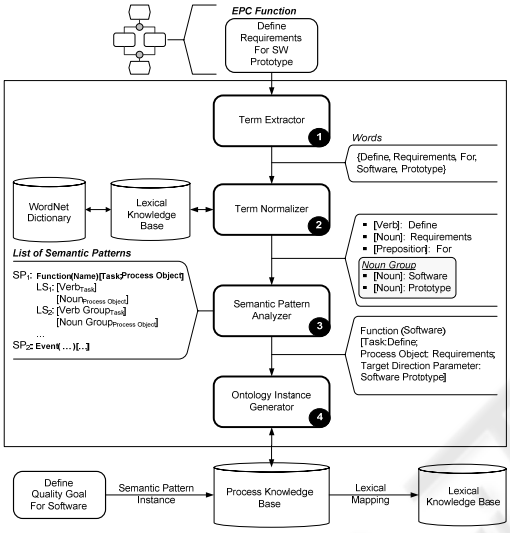

Figure 5: Stages for Automatic Semantic Annotation.

Figure 6 sketches the output of the semantic annotation process which comprises (1) a semantic linkage between EPC functions/events and instances of process ontology concepts and (2) updated instances of reference ontology concepts.

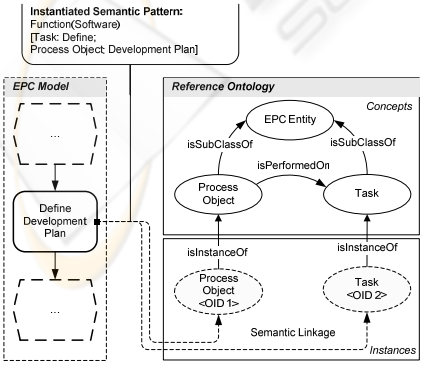

Figure 6: Example for semantically annotated EPC Function.

3.1 Term Extractor

The semantic annotation process starts with the extraction of used words by parsing natural language expressions for each EPC function/event. Extracted words are input for the term normalizer.

3.2 Term Normalizer

The term normalizer component addresses the problems of word classification and of naming conflicts. This step reduces the number of potential naming conflicts to synonyms and abbreviations. It neglects homonyms since it is assumed a non ambiguous meaning of used vocabulary in engineering domains.

Determination of word classes (e.g. noun, verb, etc.) requires finding a match between words in natural language expressions (extracted from the Term Extractor) and associated words to lexical entries in the lexical knowledge base. A match procedure considers semantic relationships (e.g. isAbbreviationTo) associated to a lexical entry (e.g. SW is an abbreviation of Software). If a search for is successful, the word class derives from the concept name the matched word is instance of. In case of naming conflicts, the term normalizer follows the rule to deliver the base word. For instance, SW has been identified as an abbreviation of software, consequently, the term normalizer delivers the term *"Software"* as a noun.

If a query for a word in the lexical knowledge base delivers an empty result, an automatically driven word classification is not feasible. In this case, the publicly available dictionary WordNet is employed for word classification and synonym detection. According to Lui and Sing (2004), it is particularly suited for this task as it is *"optimized for lexical categorization and word-similarity determination"*. WordNet originates from the Cognitive Science Laboratory of Princeton. Its schema comprises the three main classes *synset*, *wordSense* and *word*. A synset groups words with a synonymous meaning, such as {car, auto, machine}. Due to different senses of words, a *synset* contains one or more *wordsenses* and each *wordsense* belongs to exactly one *synset* (W3C, 2006). A synset either contains the word classes nouns, verbs, adjectives or adverbs. There are seventeen relations between synsets (e.g. hyponymy, entailment, meronymy, etc.) and five between word senses (e.g. antonym, see also).

The term normalizer tries to retrieve semantic information by consulting WordNet. A WordNet

query delivers either a set of word classes and synonyms (associated to the queried lexical term) or an empty set. In case of delivering an empty set the term normalizer component requires an interaction with the analyst in order to get a human classification entry. In our introduced example in Figure 5, the term normalizer identifies the lexical terms "*Define"* as a verb, "*Requirements"* as a noun, "*For"* as a preposition and "*Software Prototype"* as a noun group.

3.3 Semantic Pattern Analyzer

The term normalizer component determines word classes and resolves word conflicts as described in previous section. The semantic pattern analyzer instantiates semantic patterns by employing associated analysis rules.

Semantic pattern descriptions enable to formalize and to specify naming conventions. Naming conventions represent guidelines for naming EPC functions/events as proposed by the ARIS method and by the guidelines of modeling (Schütte, 1998).

These conventions suggest for naming EPC functions to make use of a verb to express a task followed by a noun that refers to a process object (e.g. Define: [Task] Development Plan: [Process Object]). The naming conventions for EPC events suggest expressing state information by a passive verb followed by an associated process object (e.g. Development Plan: [Process Object] Defined: [State]). The singular noun form is propagated since a process object can be regarded as a class type. Hence, the conventions proposed in data or class modeling are advocated.

A semantic pattern description is given as a tuple *S = (T, LA)* where *T* defines the semantic pattern template, *LA* is a set of pairs ($\{l_i,a_j\}$| ($L=\{l_1,...l_n\}$, $A=\{a_1,...a_n\}$) where $l_i \in L$ and $a_j \in A$) where *L* is a set of lexical structures and *A* is a set of analysis rules.

A semantic pattern template is a tuple *P=(E,C,O)* where $E \in \{Function, Event\}$ defines the semantic pattern type, *C* defines the context of a template instance, *O* is the set of addressed process ontology concepts (e.g. task, state). As an example for a semantic pattern description, following pattern template *Function(Context)[Task; Process Object]* is introduced. It is used to discuss the other parts of the semantic pattern in the further subsections.

3.3.1 Lexical Structures

The lexical structure is a tuple (I, C) where *I* is an unique identifier and *C* is an ordered set of word classes whereas the following set of predefined word classes is available: {Noun [N], NounGroup [NG], Verb [V], Passive Verb[PV], Preposition[P], Conjunction[C]}.

The introduced example for a semantic pattern description defines the following two lexical structures L_1:= [V_{Task}] [$N_{ProcessObject}$] and L_2:= [V_{Task}] [$NG_{ProcessObject}$].

3.3.2 Analysis Rules

Analysis rules evaluate natural language expressions against predefined lexical structures and instantiates one or several semantic pattern templates. If the lexical structure of a natural language expression corresponds to a lexical structure associated to a semantic pattern template, a semantic pattern template is instantiated by the assignment of lexical terms to the addressed process ontology concepts of the semantic pattern template. Consider the natural language expression *"[Verb]: Define [Noun]: Requirements"* used for naming an EPC function. It matches with the lexical structure *[$Verb_{Task}$] [$Noun_{ProcessObject}$]* of to the semantic pattern template *Function(Context)[Task; Process Object])*. A defined analysis rule for this semantic pattern template maps the lexical terms *"Define"* to the process ontology concept *Task* and *"Requirements"* to the process ontology concept *Process Object* and instantiates the semantic pattern template *Function(Software)[Task: "Define" Process Object: "Requirements"]*.

An analysis rule is specified by a precondition and a body separated by a "→". The precondition consists of the operator *Match* whose parameters represent (1) a predefined lexical structure, (2) logical expressions (e. g. Preposition="for") and (3) a list of lexical terms (E={$T_1 \ldots T_n$} or F={$T_1 \ldots T_n$}) extracted by the term normalizer. The body denotes an action that generates one or several instantiated semantic pattern templates.

Analysis rules are also used to determine the semantics of parameters. The semantics of a parameter depends on the preposition associated to a noun. For instance, the rule R: IF MATCH([V_{Task}] [$N_{processObject}$] [Parameter ="For"] [$N_{ProcessObject}$], E={$T_1 \ldots T_n$}) → GENERATE(Function (Context) [Task:V; Process Object: N; Target Direction Parameter:N]) generates an instantiated semantic pattern having a *Target Direction Parameter*. A *Source Direction Parameter* is determined by the prepositions *"FROM"* or *"OF"*, a *Target Direction Parameter* by the prepositions *"FOR"*, or "IN", a *Means Parameter* by the prepositions *"WITH",* a

Dependency Parameter by the preposition *"UPON"*.

Another feature denotes the setting of state information required for a semantic analysis of EPC events. State information indicates an attribute for a process object with an assigned attribute value. For example, the state information *"Quality Goals Defined"* assigns the attribute *"Defined"* to the process object *"Quality Goal"* with the boolean value *true*. This is realized by the rule R: IF MATCH([$N_{ProcessObject}$] [PV_{State}], E={$T_1 \ldots T_n$}) → GENERATE(Event (Context) [State:PV; State Value: "True"; Process Object: N]).

Analysis rules also play a vital role in resolving the semantics of natural language expressions that address (1) more than one task or state or (2) more than one process object or (3) more than one parameter or (4) a combination of them. To illustrate this situation, let us consider the following example: *F(Identify And Analyze Quality Goal)*. This function specifies the two tasks "*Identify"* and "*Analyze"*, connected via an *"And"* conjunction that are executed on the process object *"Quality Goal"*. For capturing the process semantics of this function in the process ontology, two semantic pattern instances are generated by applying the analysis rule R: IF MATCH([V_{1Task}] [Con="AND"] [V_{2Task}] [$NG_{ProcessObject}$], F={$T_1 \ldots T_n$}) → {GENERATE(Function(Software)[Task:V_1; Process Object:NG] , F={$T_1 \ldots T_n$}), {GENERATE (Function (Software) [Task:V_2; Process Object:NG], F). This analysis rule results in the two instantiated semantic patterns templates Function (Software) [T:"Identify"; PO:"Quality Goal"] and Function (Software) [T:"Analyze"; PO:"Quality Goal"].

3.3.3 Common Semantic Patterns

By manually performed analyzes of about 5,000 EPC functions/events in engineering domains we gained the insight that the suggested naming conventions do not fully cover the implicit process semantics of natural language expressions used for naming EPC functions/events. As an additional contribution of this paper, we introduce a set of semantic pattern descriptions resulting from our investigations. The Figures 7-10 summarize these semantic pattern descriptions for EPC functions/events. They use the following set of abbreviations: {V:=Verb, N:=Noun, NG:=Noun Group, PO:=Process Object, T:=Task, Con:=Conjunction, P:=Preposition, F:=Function, C:=Context, TDP:=Target Direction Parameter, SDP:=Source Direction Parameter, MP:=Means Parameter, DP:=Dependency Parameter}.

Lexical Structure: LS_1 := [V] [N] Analysis Rule: IF MATCH([V] [N], F) → GENERATE(F(C)[T:V; PO:N]	Example: F (Define Goal): Pattern Template Instance: SP_1 := Function(Software) [T:"Define"; PO: "Goal"]
Lexical Structure: LS_2 := [V] [NG] Analysis Rule: IF MATCH([V] [NG] , F) → GENERATE(F(C)[T:V; PO:NG]	Example: F (Define Quality Goal): Pattern Template Instance: SP_1 := Function(Software) [T:"Define"; PO: "Quality Goal"]
Lexical Structure: LS_3 := [V_1] [C] [V_2] [N] Analysis Rule: IF MATCH([V_1] [Con="AND"] [V_2] [N] , F) → { GENERATE (F(C)[T:V_1; PO:N], GENERATE (F(C)[T:V_2; PO:N]}	Example: F (Identify And Analyze Goal): Pattern Template Instance: SP_1 := Function(Software) [T:"Identify"; PO: "Quality Goal"] SP_2 := Function(Software) [T:"Analyze"; PO: "Goal"]
Lexical Structure: LS_4 := [V] [NG_1] [C] [NG_2] Analysis Rule: IF MATCH([V] [NG_1] [Con="AND"] [NG_2] , F) → { GENERATE (F(C)[T:V; PO:NG_1], GENERATE (F(C)[T:V; PO:NG_2]}	Example: F (Define Quality Goal And Quality Measure): Pattern Template Instance: SP_1 := Function(Software) [T:"Define"; PO: "Quality Goal"] SP_2 := Function(Software) [T:"Define"; PO: "Quality Measure"]

Figure 7: Semantic Pattern Description for the Template Function(Context)[Task; Process Object].

Lexical Structure: LS_4 := [V] [NG_1] [P] [NG_2] Analysis Rule: IF MATCH([V] [NG_1] [P=("FROM" | "OF")] [NG_2] , F) → { GENERATE (F(C)[T:V; PO:NG_1; *SDP:*NG_2]}	Example: F (Derive Quality Goal From Specification Document) Pattern Template Instance: SP_1 := Function(Software) [T:"Derive"; PO: "Quality Goal", *SDP: "Specification Document"*]
Lexical Structure: LS_4 := [V] [NG_1] [P] [NG_2] Analysis Rule: IF MATCH([V] [NG_1] [P=("FOR" | "ON" | "IN") , F] [NG_2]) → { GENERATE (F(C)[T:V; PO:NG_1; *TDP:*NG_2]}	Example: F (Define Quality Goal For Project Plan) Pattern Template Instance: SP_1 := Function(Software) [T:"Define"; PO: "Quality Goal", *TDP: "Project Plan"*]
Lexical Structure: LS_4 := [V] [N_1] [P] [N_2] Analysis Rule: IF MATCH([V] [N_1] [P="WITH"] [N_2] , F) → { GENERATE (F(C)[T:V; PO:N_1; *MP:*N_2]}	Example: F(Rework Specification With Customer) Pattern Template Instance: SP_1 := Function(Software) [T:"Rework"; PO: "Specification", *MP: "Customer"*]
Lexical Structure: LS_4 := [V] [NG_1] [P] [NG_2] Analysis Rule: IF MATCH([V] [NG_1] [P="UPON"] [NG_2] , F) → {GENERATE (F(C)[T:V; PO:NG_1; *DP:*NG_2]}	Example: F(Decide Quality Measure Upon Review Status) Pattern Template Instance: SP_1 := Function(Software) [T:" Decide"; PO: "Quality Measure", *DP: "Review Status"]*

Figure 8: Semantic Pattern Description for the Template Function(Context)[Task; Process Object; Parameter].

Lexical Structure: LS_1 := [N] [PV] Analysis Rule: IF MATCH([N] [PV] , E) → GENERATE(E(C)[PO:N; S:PV]	Example: E (Goal Defined): Pattern Template Instance: SP_1 := Event(Software) [PO: "Goal"; S:"Defined"]
Lexical Structure: LS_2 := [NG] [PV] Analysis Rule: IF MATCH([NG] [PV], E) → GENERATE(E(C)[PO:NG; S:PV],	Example: E (Quality Goal Defined): Pattern Template Instance: SP_1 := Event(Software) [PO: "Quality Goal"; S:"Defined"]
Lexical Structure: LS_3 := [N] [PV_1] [C] [PV_2] Analysis Rule: IF MATCH([N] [PV_1] [Con="AND"] [PV_2], E) → { GENERATE (E(C)[S:PV_1; PO:N], GENERATE (E(C)[S:PV_2; PO:N]}	Example: E (Goal Identified And Analyzed): Pattern Template Instance: SP_1 := Event(Software) [PO: "Goal"; S:"Identified"] SP_2 := Event(Software) [PO: "Goal"; S:"Analyzed"]
Lexical Structure: LS_4 := [NG_1] [C] [NG_2] [PV] Analysis Rule: IF MATCH([NG_1] [Con="AND"] [NG_2] [PV], E) → {GENERATE(E(C)[PO:NG_1; S:PV], GENERATE(E(C)[PO:NG_2; S:PV;]}	Example: E (Quality Goal And Quality Measure Defined): Pattern Template Instance: SP_1 := Event(Software) [PO: "Quality Goal"; S:"Defined"] SP_2 := Event(Software) [PO: "Quality Measure"; S:"Defined"]

Figure 9: Semantic Pattern Description for the Template Event(Context)[Process Object; State].

Lexical Structure: LS_4 := [NG_1] [P] [NG_2] [PV] Analysis Rule: IF MATCH([NG_1] [P=("FROM" | "OF")] [NG_2] [PV], E) → { GENERATE (E(C)[PO:NG_1; *SDP:NG*$_2$; S:PV]}	Example: E (Quality Goal From Specification Document Derived) Pattern Template Instance: SP_1 := Event(Software) [PO: "Quality Goal", *SDP: "Specification Document"*; S:"Derived"]
Lexical Structure: LS_4 := [NG_1] [P] [NG_2] [PV] Analysis Rule: IF MATCH([NG_1] [P=("FOR" | "ON" | "IN")] [NG_2] [PV] , E) → { GENERATE (E(C)[PO:NG_1; *TDP:NG*$_2$; S:PV]}	Example: E (Quality Goal For Project Plan Defined) Pattern Template Instance: SP_1 := Event(Software) [PO: "Quality Goal", *TDP: "Project Plan"*; S:"Defined"]
Lexical Structure: LS_4 := [N_1] [P] [N_2] [PV] Analysis Rule: IF MATCH([N_1] [P="WITH"] [N_2] [PV] , E) → { GENERATE (E(C)[PO:N_1; *MP:N*$_2$; S:PV]}	Example: E(Specification With Customer Reworked) Pattern Template Instance: SP_1 := Event(Software) [PO: "Specification", *MP: "Customer"*; S:"Reworked"]
Lexical Structure: LS_4 := [NG_1] [P] [NG_2] [PV] Analysis Rule: IF MATCH([NG_1] [P="UPON"] [NG_2] [PV], E) → {GENERATE (E(C)[PO:NG_1; *DP:NG*$_2$; S:PV]}	Example: E(Quality Measure Upon Review Status Decided) Pattern Template Instance: SP_1 := Event(Software) [PO: "Quality Measure", *DP: "Review Status"*; S:" Decided"]

Figure 10: Semantic Pattern Description for the Template Event(Context)[Process Object; State; Parameter].

3.4 Ontology Instance Generator

The ontology instance generator populates the reference ontology by adding or updating instances of predefined concepts and relations in the lexical and in the process knowledge base. Instantiated semantic patterns generated by the semantic pattern analyzer are input for the ontology instance generator. This step concludes with the establishment of the semantic linkage between EPC functions/events and concerned reference ontology instances.

4 RELATED WORK

The work presented in this paper refers to research activities involving semantic Business Process Management and Natural Language Processing. Enhancing Business Process Management (BPM) with semantic web technologies to overcome obstacles in automated processing has triggered a new wave in research and practice (e.g. Hepp et al., 2005, Hepp et al., 2007). Process ontology design is a well-established field of research consisting of many distinguished approaches. The most important are: Business Process Management Ontology (BPMO) is a fully-fledged semantic business process modeling framework (Yan et al., 2007). Semantic EPC (sEPC) (Hepp et al., 2007) has emerged from the SUPER Project (Super, 2007) and aims at supporting the annotation of EPC models. Thomas and Fellmann (2007) describe a similar approach that addresses the semantic annotation of EPC models. Plan ontologies such as the Dolce+DnS Plan Ontology (DPPO) (Gangemi et al., 2004) are founded on a theory of planning problems and on semantic descriptions of plans.

The process ontology proposed in this paper prelimary intends to capture the implicit semantics of natural language expressions used for naming EPC functions/events (e.g. relationships between tasks and process objects, state information resulting from performing a task). Further, the introduced process ontology does not consider the control flow.

The semantic annotation of EPC models is comparable with approaches used by Semantic Web annotation platforms (SAPs) whose purpose is to annotate existing and new documents on the Web. SAPs can be classified according to the used annotation method. The two primary categories are Pattern-based and Machine Learning-based. In Pattern-based approaches, *"an initial set of entities is defined and the corpus is scanned to find the*

patterns in which the entities exist." Reeve and Han (2005). Machine learning-based SAPS utilize probability and induction methods.

The approach presented in this paper follows the paradigm of a pattern-based analysis of natural language expressions used to describe EPC functions/events by employing semantic patterns. The idea using semantic patterns is inspired by Rolland and Achour (1998). They employ semantic patterns to extract a use case model from ambigous textual use case descriptions. Further, the usage of patterns traces back to Hearst (1992). The underlying clue is the use of patterns whose purpose is to explicitly grasp a certain relation between words. (Biemann, 2005), (Cimiano et al., 2006).

In this work, instantiated semantic patterns bridge the gap between informal and formal representations. Instances of predefined semantic patterns establish the semantic linkage between EPC functions/events and instances of process ontology concepts.

5 CONCLUSIONS

The introduced approach shows how to perform an automated semantic annotation of EPC functions/events. It employs semantic pattern descriptions to bridge the gap between semi-formal process representations and formal reference ontologies. Semantic pattern descriptions allow the specification of semantic pattern templates (naming conventions for EPC functions/events), lexical structures (grammar of natural language expressions) and analysis rules (instantiation of semantic pattern templates).

Our proposal for an automated semantic annotation is limited for resolving the semantics of natural language expressions used for a description of EPC functions/events. These natural language expressions must obey basic naming conventions as suggested by the EPC modeling language. A task within an EPC function is expressed by means of a verb, state information is indicated by a passive verb. Despite these limitations, the declarative nature of semantic pattern descriptions enables to define an arbitrary set of naming conventions. The definition of semantic pattern descriptions provides a mechanism to standardize the naming of EPC functions/events in a distributed modeling environment. The proposed common semantic patterns in section 3.3.3 resulted from practical experiences gained on a human driven analysis of about 5,000 EPC functions/events in engineering domains. The identified semantic pattern descriptions are a first approach for an additional standardization concerning the naming EPC functions/events.

A semantic annotation of EPC models yields several advantages. A resulting reference ontology represents a necessary prerequisite for the identification of patterns (structural analogies) in process models as proposed by Schütte (1998, 237). The frequency of occurrence of process patterns enables an objective measure to evaluate candidates for common or best practice solutions. The dependencies between patterns can provide information on larger structures (reference models, Schermann et al., 2007) or process variants (configurable and generic adaptation of reference models, Becker et al., 2007).

ACKNOWLEDGEMENTS

Thanks to Mathias Goller for the fruitful discussions and for proofreading this paper.

REFERENCES

Becker, J. et al. (2007). Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Becker J.; Delfmann, P. (Ed.): *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*. Physica, Heidelberg, p. 27-58.

Biemann, C. (2005). Ontology Learning from Text: A Survey of Methods. *LDV-Forum* 20 (2), 75-93.

Bögl, A. et al. (2008). Knowledge Acquisition from EPC Models for Extraction of Process Patterns in Engineering Domains. *In Proceedings der Multikonferenz Wirtschaftsinformatik*, (MKWI 2008), München, Deutschland.

Cimiano, P. et al. (2006). Ontologies on Demand? – A Description of the State-of-the-Art, Applications, Challenges and Trends for Ontology Learning from Text. *Information, Wissenschaft und Praxis* 58 (6-7): 315-320. October 2006.

Gangemi, A. et al. (2005). Task Taxonomies for Knowledge Content D07, www.loa-cnr.it/Papers/D07_v21a.pdf, 13.09.2007.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. *In Proceedings of the 14th International Conference on Computational Linguistics* (COLING 1992), Volume 2, Nantes, France, pp. 539-545.

Hepp, M. et al. (2005). Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. IEEE

International Conference on e-Business Engineering. Beijing, China, p. 535-540.

Hepp, M. et al. (ed.) (2007). *Proceedings on Semantic Business Process and Product Lifecycle Management*, 3rd European Semantic Web Conference. Innsbruck, Austria.

Keller, G. et al. (1992). Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“ In: Scheer, A-W. (Hrsg.): *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Heft 89, Saarbrücken, http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf, 20.08.2007

Liu, H.; Singh, P. (2004). ConceptNet – A Practical Commonsense Reasoning Tool-Kit, *BT Technology Journal,* (22) 4, p. 211-226.

Moore, J. et al. (2000). Combining and Adapting Process Patterns for Flexible Workflow. 11th International Conference on Database and Expert Systems Applications, London, United Kingdom, p. 797-801.

Pfeiffer, D.; Gehlert, A. (2005). A Framework for Comparing Conceptual Models. Workshop on Enterprise Modelling and Information Systems Architectures. Klagenfurt, Austria, p. 108-122.

Reeve, L.; Han, H. (2005). Survey of semantic annotation platforms. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, Santa Fe, New Mexico, March 13 - 17, 2005.

Roberto Basili et al. (2005). Language Learning and Ontology Engineering: an Integrated Model for the Semantic Web. 2nd Meaning Workshop, Trento, Italy, February 2005.

Rolland, C.; Achour Ben, C. (1998). Guiding the construction of textual use case specifications, in the *Data & Knowledge Engineering Journal*, 25(1-2) Special Jubilee issue, March 1998, 125-160.

Rosemann, M. (1996). Komplexitätsmanagement in Prozeßmodellen. Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung. Gabler, Wiesbaden.

Schermann, M. et al. (2007). Fostering the Evaluation of Reference Models: Application and Extension of the Concept of IS Design Theories. 8th International Conference Wirtschaftsinformatik. Karlsruhe, Germany, p. 181-198.

Schütte, R. and T. Totthowe, T. (1998). The Guidelines of Modeling as an approach to enhance the quality of information models. In: *Conceptual Modeling* - ER '98. 17th International ER-Conference, Singapore, November 16-19, 1998. T. W. Ling, S. Ram, M. L. Lee (Eds.), pages 240-254. Berlin.

Schütte, R. (1998). Grundsätze ordnungsgemäßiger Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle. Gabler, Wiesbaden.

Super, Integrated Project Semantics Utilized for Process Management within and between Enterprises. http://www.ip-super.org, 13.09.2007.

Thomas, F., Fellmann, M. (2007). Semantic Business Process Management: Ontology Based Process Modeling Using Event-Driven Process Chains. *International Journal of Interoperability in Business Information Systems*, (2) 1, p. 29-44..

W3C (2006). RDF/OWL Representation of WordNet, W3C Working Draft June 2006, http://www.w3.org/TR/wordnet-rdf/, 13.09.2007.

Yan, Z. et al. (2007). BPMO : Semantic Business Process Modeling and WSMO Extension. International Conference on Web Services. Salt Lake City, USA, p. 1185-1186.