Semantic Enabled Complex Event Language for Business Process Monitoring

Dong Liu
Knowledge Media Institute
The Open University
Walton Hall, Milton Keynes, UK
d.liu@open.ac.uk

Carlos Pedrinaci
Knowledge Media Institute
The Open University
Walton Hall, Milton Keynes, UK
c.pedrinaci@open.ac.uk

John Domingue
Knowledge Media Institute
The Open University
Walton Hall, Milton Keynes, UK
j.b.domingue@open.ac.uk

ABSTRACT
Efforts are being made to enable business process monitoring and analysis through processing continuously generated events. Several ontologies and tools have been defined and implemented to allow applying general-purpose Business Process Analysis techniques to specific domains. On this basis, a Semantic Enabled Monitoring Event Language (SEMEL) is proposed to facilitate defining complex queries over monitoring data so as to interleave temporal and ontological reasoning. In this paper, the formal semantics of SEMEL is discussed, and the implementation approach of SEMEL interpreter is also briefly described, which encompasses translation into an operational language.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Contructs and Features – *constraints, data types and structures, patterns.*

General Terms
Management, Documentation, Design, Languages.

Keywords
Semantics, Complex Event Language, Event Pattern, Business Process, Monitoring.

1. INTRODUCTION
Business Process Analysis uses the logs generated by systems such as Workflow Engines or Enterprise Resource Planning (ERP) solutions in order to, on the one hand, track the execution of processes and identify potential improvements, and on the other hand, verify and validate the actual execution of processes with respect to prescribed or expected behaviour. We have previously argued for the need for applying general purpose analysis techniques over specific domains in a way that allows analysts to use their particular terminology and existing knowledge about their domain and we have defined and implemented a set of ontologies and tools to cater for this [1]. In this paper we propose SEMEL, a Semantic Enabled Complex Event Language for Business Process Monitoring that provides an additional layer of abstraction allowing the definition of complex queries over monitoring data interleaving temporal and ontological reasoning, through an easy-to-use SQL-like language.

We first describe an ontology-based event model, on the basis of which SEMEL is designed, and then follow up specification of syntax and formal semantics, as well as examples for basic usages of SEMEL. Finally, we briefly describe the implementation approach of SEMEL interpreter.

2. ONTOLOGY-BASED EVENT MODEL
Business monitoring events, which are made up of timestamp, causality, and a set of attribute values, signify and record runtime behaviours and execution histories of business processes. In order to describe semantics of monitoring events and automatically reason over them, we adopted an ontology-based event model, which consists of three pre-defined ontologies: Core Ontology for Business pRocess Analysis (COBRA), Event Ontology (EVO) and Time Ontology [1].

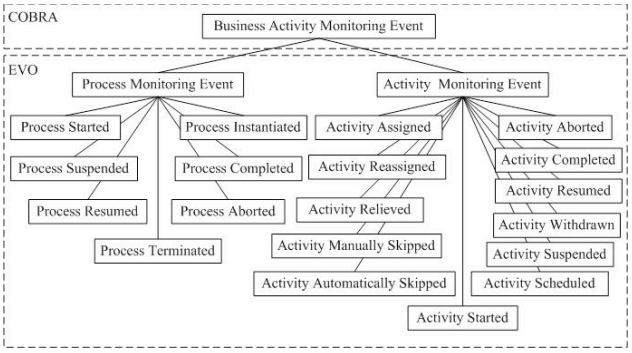

Figure 1. EVO and Part of COBRA Ontology [1]

COBRA provides a core terminology for business process monitoring and analysis, which includes the *Monitoring Event* concept. *Monitoring Event* characterized by a timestamp, a causality vector and associated data, is the common ancestor of all monitoring events.

EVO extends COBRA ontology (see Figure 1) by a set of concepts to capture states and transitions of process or activity instances, namely seven *Process Monitoring Events* and twelve *Activity Monitoring Events*. For instance, *Activity Started* concept in EVO implies that a new instance of activity is created.

Time Ontology sets forth the timing properties of events and temporal reasoning. Time Ontology defines three top-level concepts, namely *Time Instant*, *Time Interval* and *Temporal Entity*, and it implements both the interval relations defined in Allen's interval algebra [2] and additional instant-interval relations see Figure 2.

Since the ontology-based event model serves as the foundation of SEMEL language, atomic and complex events are respectively defined as follows.

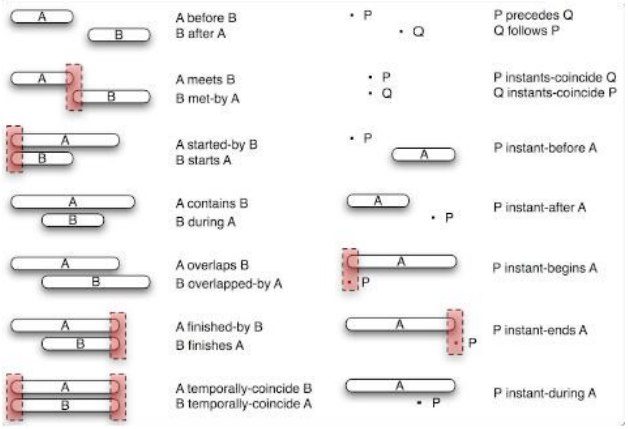

Figure 2. Temporal Relations [1]

- **Definition 1 (Atomic Event).** An atomic event, denoted by lower-case letter *e*, is defined to be an instance of concept *Monitoring Event* in COBRA ontology or its sub-concepts. As an event refers to an instantaneous occurrence of interest, a particular attribute of the concept *Monitoring Event*, known as "*occurAt*", is used to specify the time when the event happens.
- **Definition 2 (Complex Event).** Complex events are built up by a set of atomic events, which hold certain temporal relationships or satisfy constraint conditions on attributes. The occurrence time of a complex event is an interval rather that a time instant, which starts when its earliest constituent event happens, and ends when the latest constituent happens.

3. SEMEL LANGUAGE

As stated in [3], an event language should satisfy four primary requirements, i.e. power of expressions, notational simplicity, precise semantics, and scalable pattern matching. Besides the syntax and semantics, in this section, we will also clarify that the proposed SEMEL language can satisfy these requirements to a certain extent.

3.1 Syntax and Structure

SEMEL is a declarative language with a SQL-like syntax similar to [4], which has EVENT, FROM, WHERE and WITHIN clauses (see the listing below). These four clauses respectively depict event patterns, event source, attribute constraints and a sliding window of observation. By this means, the creation of ontological expressions for specifying complex events is simplified, and the similarity with SQL helps reducing the learning curve while retaining the expressivity power.

```
EVENT <event_patterns>
[FROM <event_sources> ]
[WHERE <attribute_constraints> ]
[WITHIN <sliding_window> ]
```

The EVENT clause in a SEMEL statement specifies event patterns to be detected during event processing, which will be detailed in Section 3.2. The FROM clause indicates sources of events to be queried on, e.g. the log repositories of workflow engines. The existing event languages such as [4] only support value-based constraints. In contrast, the WHERE clause of SEMEL allows the conditions being expressed ontologically. In this way, we can benefit from ontology reasoning and also seamlessly integrate domain specific knowledge within SEMEL. The WHERE clause, together with FROM clause makes up the non-temporal part of the specification in SEMEL language. The BNF grammar of event source and attribute constraint is:

```
<event_sources>        ::= <source_name> { <source_name>}
<attribute_constraints> ::= <constraint> {<conj> <constraint>}
<constraint>           ::= <attribute_name> "(" <event_name>
                           ")" <comp> <attribute_value>
<attribute_value>      ::= <const> | <attribute_name> "("
                           <event_name>")"
<conj>                 ::= and | or | not
<comp>                 ::= "<" | ">" | "=" | "<=" | ">="
```

The WITHIN clause imposes a time bound on the collection of events by an interval, and events happen outsides will not be taken into account. The WITHIN and EVENT clauses are the temporal parts of SEMEL. Herein is the BNF of the sliding window of observation, which is introduced by WITHIN clause:

```
<sliding_window> ::= <integer> <time_unit>
<time_unit>      ::= second | minute | hour | day | month | year
```

The rest of this section will detail the specification of event pattern, and exemplify the basic usage. Additionally, the formal semantics of SEMEL language will be put forward.

3.2 Event Pattern Specification

Event patterns are built by event constructs. The existing event languages vary in their supports to event constructs, especially the negation construct [5]. In SEMEL language, event constructs can be divided into two classes: primitive constructs and composite constructs. The former ones are comprised of temporal constructs and negation constructs. Temporal constructs correspond to the temporal relationships defined in Time Ontology, i.e. "precedes", "follows", "before", "meets", "during", etc. Negation constructs mean the event never happens during the given time period.

Composite constructs are combinations of the primitive ones, in other words, if we regard event constructs as predicates on monitoring data, composite constructs will be composite predicates. For example, the "sequence" can be defined as:

$$sequence\left(e_1, e_2, \cdots, e_n\right) = before\left(e_1, before\left(e_2, \cdots, before\left(e_{n-1}, e_n\right)\right)\right)$$

The BNF grammar of event pattern specification in SEMEL is as follows:

```
<event_patterns>   ::= <atomic_event> |
                       <event_construct>(<event_list>)
<event_list>       ::= <event> { "," <event>}
<event>            ::= <atomic_event> | <complex_event>
<complex_event>    ::= <atomic_event> |
                       <event_construct>"("<event_list>")"
<atomic_event>     ::= <event_name>":"<event_concept>
<event_construct>  ::= <t_construct> | <n_constrct> |
                       <c_construct>
<t_construct>      ::= precedes | follows | before | meets |
                       meets | during | after | starts | finishes |
                       overlaps | instantsCoincide |
                       temporallyCoincide
<n_construct>      ::= never | not
<c_construct>      ::= sequence
```

3.3 Semantics

Because the underlying formalism of the adopted COBRA, EVO and Time Ontology is Description Logics, and temporal relations are binary logical relations between individuals, we give the semantics of SEMEL following the model-theoretic way of DLs [6]. Formally, the semantics of DLs is revealed by a pair $(\Delta^I, \bullet^I)$, which is also known as an interpretation. Δ^I is a set of individuals, while $\bullet^I$ maps every concept to a sub-set of Δ^I, every attribute to a sub-set of $\Delta^I \times \Delta^I$ as well.

Overall, temporal constructs are a set of predicates on $C^I \times D^I$, and *C*, *D* are sub-concepts of *Monitoring Event*, i.e. $C^I, D^I \subseteq MonitoringEvent^I$. The semantics of construct "*before*" is shown below, and those of all other temporal constructs can also be defined in the same way.

$$before(e:C, f:D) = \{e \in C^I, f \in D^I \mid \exists t_1, t_2 \in TimeInstance^I, (e,t_1) \in occurAt^I \wedge (f,t_2) \in occurAt^I \wedge (t_1,t_2) \in before^I\}$$

Here, $before^I$ is the interpretation of the corresponding temporal relation "before" defined in the aforementioned EVO ontology, which is essentially a binary relation on the interpretation of *Time Instance*, i.e. $before \subseteq TimeInstance^I \times TimeInstance^I$

The negation constructs, "never" and "not", are predicates on $C^I \times TimeInstance^I \times TimeInstance^I$. *C* is also one of the sub-concepts of *Monitoring Event*, i.e. $C^I \subseteq MonitoringEvent^I$, thus, the semantics of "never" is:

$$never(e:C, st, et) = \{e \in C^I \mid \neg(\exists t \in TimeInstance^I, (e,t) \in occurAt^I \wedge (st,t) \in precedes^I \vee (t,et) \in precedes^I)\}$$

Ternary predicate "never" can be used on its own. In contrast, the other negation construct "not", which is a unitary predicate, should arise inside the construct "sequence". Additionally, the semantic of "not" is similar to that of "never", but the starting and ending time are determined by the other events of sequence or observation window.

The attribute constraints filter monitoring events by certain conditions. For example, *causedBy* selects all the events caused by a given event, i.e.

$$causedBy(e) = \{f \in MonitoringEvent^I \mid (f,e) \in causedBy^I\}$$

As for the WITHIN clause, it filters the constituents of a complex event by a time interval. Supposed that e_s, e_e, having occurrence time t_s, t_e, are respectively the earliest and latest constituent event of a pattern, then clause "WITHIN t" means $t_e - t_s \le t$.

3.4 Examples

In this section, we illustrate the basic usages of SEMEL language by the following examples:

1. An activity instance completes within 10 minutes:

```
EVENT sequence(e1:ActivityStarted,e2:ActivityCompleted)
WHERE concernsActivityInstance(e1)
        = concernsActivityInstance(e2)
WITHIN 10 minute
```

2. The execution time of an activity instance exceeds 20 minutes:

```
EVENT sequence(e1:ActivityStarted),
        not(e2:ActivityMonitoringEvent))
WHERE concernsActivityInstance(e1)
        = concernsActivityInstance(e2)
WITHIN 20 minute
```

3. Three instances of activity act1 start within 1 minutes

```
EVENT sequence(e1:ActivityStarted, e2:ActivityStarted,
                e3:ActivityStarted)
WHERE  performs(concernsActivityInstance(e1))=act1 and
        performs(concernsActivityInstance(e2))=act1 and
        performs(concernsActivityInstance(e3))=act1
WITHIN 1 minute
```

4. IMPLEMENTATION

A SEMEL interpreter will be implemented by translating it into the Operational Conceptual Modeling Language (OCML) [7], which provides support for executing the definitions in the ontology and export mechanisms to other representations such as OWL and WSML. The translation starts with the declaration of variables, each of which will be translated into an "instance-of" clause. Let "(e:Concept)" be a declaration of event *e*, the translation result of it will be "(instance-of ?e iri)". Here, "iri" represents the IRI of the designated event concept.

Temporal constructs will be translated to an "and" clause. For instance, the "before" construct will be restated in OCML as following:

```
(and (instance-of ?t1 #_TIME:TimeInstant)
     (instance-of ?t2 #_TIME:TimeInstant)
     (has-slot-value ?e1 #_EVO:occurAt ?t1)
     (has-slot-value ?e2 #_EVO:occurAt ?t2)
     (#_TIME:before ?t1 ?t2) )
```

Translation of other temporal constructs will be performed in the same way. Since the composite event constructs are combinations of the primitive ones, they can recursively be translated into OCML segments. Negation constructs are mapping to a "not-exist" clause in OMCL. For instance, the translation result of "not" construct in second example shown in Section 3.4 is presented below, where values of t1, t2 are determined by event e1 and the WITHIN clause.

```
(and (instance-of ?t1 #_TIME:TimeInstant)
     (instance-of ?t2 #_TIME:TimeInstant)
     (not (exists ?e2 (and (instance-of ?e2
       #_EVO:ActivityMonitoringEvent)
     (has-slot-value ?e2 #_EVO:occurAt ?oe2)
     (#_TIME:before ?t1 ?oe2)
     (#_TIME:before ?oe2 ?t2)))))
```

We explain translation of the WHERE clause by an example, namely the first one shown in Section 3.4, which is translated as:

```
(and (instance-of ?e1
       #_EVO:BusinessMonitoringEvent)
     (instance-of ?e2
       #_EVO:BusinessMonitoringEvent)
     (has-slot-value ?e1
```

```
    #_EVO:concernsActivityInstance ?cai1)
  (has-slot-value ?e2
    #_EVO:concernsActivityInstance ?cai2)
  (= ?cai1 ?cai2) )
```

Before processing the WITHIN clause such as the one of the first example in Section 3.4, we convert the time unit to millisecond, and then translate it as:

```
(and (instance-of ?oe1 #_TIME:TimeInstant)
     (instance-of ?oe2 #_TIME:TimeInstant)
     (has-slot-value ?e1 #_EVO:occurAt ?oe1)
     (has-slot-value ?e2 #_EVO:occurAt ?oe2)
     (<= (- ?oe2 ?oe1) n)) )
```

5. RELATED WORK

There are several complex event languages have been proposed in previous works, of which Cayuga, SEL, SASE, RAPIDE and EPL are the most representative ones. However, all these existing event languages do not have ontology-based event model or support to processing semantics of events.

Cayuga Event Language (CEL) is based the Cayuga Algebra and designed to query over event streams [8]. CEL takes temporally ordered sequences of tuples as the data model, and makes a simple mapping between the operators of Cayuga Algebra and a SQL-like syntax.

SASE is a declarative language with SQL-like syntax, which can be used to filter, correlate and transform events [4, 9]. Infinite sequence of events that are composed of a timestamp, the name of type and some associate attributes, serves as the underlying data model of SASE. Rather than a generalized event language, SASE specially applies to RFID-enabled applications.

SEL focus on the specification of event patterns, and takes into account the appropriateness and completeness of event operators, effectiveness and efficiency of expressions, flexibility and readability of language [5]. Especially, SEL proposes a novel way to deal with negation operator in event language.

RAPIDE event pattern language [3] is a strong-typing declarative computing language, which provides built-in data types, basic event patterns, pattern operators, temporal operators, etc. The syntax of RAPIDE is similar to the object-oriented programming languages such as C# and Java.

EPL, which stands for Event Processing Language, is the SQL-like event language of Esper—an event stream and complex event processor for Java [10]. EPL can help retrieving information from event streams, and also manipulating the event streams.

6. CONCLUSIONS AND FUTURE WORK

SEMEL, an event language for business processing monitoring and analysis is proposed in this paper. It has not only SQL-like syntax that is easy to use, but also formal semantics. SEMEL interpreter can be implemented by recursively translating SEMEL statements into the Operational Conceptual Modeling Language (OCML) [7], which provides support for executing the definitions in the ontology and export mechanisms to other representations. Our future works will also include optimization of SEMEL queries and the mechanism of automatically triggering management actions in a scalable SOA environment, when specific events happen.

7. ACKNOWLEDGMENTS

This work was funded by the European projects SUPER (FP6-026850) and SOA4All (FP7-215219).

8. REFERENCES

[1] Pedrinaci C., Domingue J., de Medeiros A. K. A. 2008. A Core Ontology for Business Process Analysis. In Proceedings of the 5th European Semantic Web Conference (Tenerife, Spain, June 01 - 05, 2008). ESWC '08. Springer Verlag, 49-64.

[2] Allen, J.F. 1983. Maintaining knowledge about temporal intervals. Communications of the ACM. 26,11, 832–843.

[3] Luckham, D. 2002. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley.

[4] Wu, E., Diao, Y., Rizvi, S. 2006. High-Performance Complex Event Processing over Streams. In Proceedings of the ACM SIGMOD International Conference on Management of Data (Chicago, IL, USA, June 26-29,2006). ACM Press, New York, NY, 407 - 418. DOI= http://dx.doi.org/10.1145/1142473.1142520

[5] Zhu, D., Sethi, A. S. 2001. SEL, a New Event Pattern Specification Language for Event Correlation. In Proceedings the 10th International Conference on Computer Communications and Networks (Piscataway, NJ, USA, October 15 - 17, 2001). ICCCN '01. IEEE Computer Society, 586-589. DOI= http://dx.doi.org/10.1109/ICCCN.2001.956327

[6] Baader, F., Burckert, H. J., Heinsohn, J., Hollunder, B., Muller, J., Nebel, B., Nutt, W. and Protlich, H. J. 1990. Terminological knowledge representation: a proposal for a terminological logic. Technical memo TM-90-04, DFKI, Saarbrucken, Germany.

[7] Motta, E. 1999. Reusable Components for Knowledge Modeling. Case Studies in Parametric Design Problem Solving. Volume 53 of Frontiers in Artificial Intelligence and Applications. IOS Press

[8] Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M. and White, W. 2007. Cayuga: a high-performance event processing engine. In Proceedings of ACM SIGMOD International Conference on Management of Data (Beijing, China, June 12 - 14, 2007). ACM, Beijing, China, 412 - 423. DOI=http://doi.acm.org/10.1145/1247480.1247620

[9] Gyllstrom, D., E. Wu, et al. 2007. SASE: Complex Event Processing over Streams. In Proceeding of the 3rd Biennial Conference on Innovative Data Systems Research (Asilomar, California, USA, January 7 - 10, 2007). CIDR '07.

[10] EsperTech Inc. 2008. Event Stream Intelligence. http://www.espertech.com/index.php