Complex Events Specification for Properties Validation

Antonia Bertolino, Antinisca Di Marco, Francesca Lonetti

Complex Events Specification for Properties Validation

Antonia Bertolino
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
{antonia.bertolino}@isti.cnr.it

Antinisca Di Marco
Computer Science Department, University of L'Aquila, Coppito (AQ), Italy
antinisca.dimarco@univaq.it

Francesca Lonetti
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
{francesca.lonetti}@isti.cnr.it

***Abstract*—Run-time validation of non-functional properties becomes very important to evaluate and keep under control dynamic and evolving systems. Event-driven monitoring is a commonly adopted approach for observing and analyzing that these properties are satisfied. As the events to be observed become more and more complex, a powerful events specification language is needed. In this paper we present a complex events specification language that is included into the Property Meta-Model (PMM). It is intuitive and easy to use and at the same time machine-processable, thus allowing for the automated run-time configuration of a model-driven event-based monitoring system. The PMM complex events specification language combines features of two existing and well-known event specification languages that are GEM and Drools Fusion, and in addition presents new features not included in the considered languages. As a proof of concept we present how the PMM complex events specification language can be used for modeling complex events excerpted from the scenarios of the CONNECT European Project.**

I. Introduction

In evolving software systems, the capability of validating non-functional properties at run-time becomes more and more important. Due to the need of adaptation, many validation and verification activities are automated and continued at run-time, by means of flexible and dynamic monitoring infrastructures. Therefore, a machine-processable specification of the properties that systems must guarantee becomes an essential asset for enacting the monitoring and testing activities.

In [9] we introduced a generic and flexible Property Meta-Model (PMM) defining the abstract structure of non-functional properties that span over dependability, performance, security and trust. On top of the PMM we also defined automated procedures (in form of Model2Code transformations) [14], [4] that, starting from a PMM model, can instrument the event-based GLIMPSE monitor [5] for run-time validation and verification of non-functional properties.

We follow a model-driven paradigm that concerning the specification of properties allows for dealing separately with their structuring and the inclusion of domain-specific concepts belonging to the application under exam. Thus in PMM we separate the property definition from the application domain and its specific ontology. We introduce the *EventType* entity that includes the terms of the application-domain ontology. The *EventType* models an observable system behavior that can be a primitive/simple event or a composite/complex event that is a combination of primitive and other composite events. With regard to the latter, however, the properties to be validated can involve events that become more and more complex. Therefore, a powerful complex events specification language is needed.

Goal of this paper is to present a comprehensive and machine processable specification language that allows for defining complex events models involved into non-functional properties. The proposed language is specified as part of our more general PMM, but it can be used in isolation to specify events that are not necessarily tied to a property modeling. Moreover, it can be adopted as a specification language independently from model-driven development approaches.

As we will discuss in the next section, our proposal for this specification language followed a survey of existing ones, through which we identified and merged into a common specification the available features from existing complex events specification languages. The proposed language further improves on them by adding new desirable features not included in the existing languages. In addition, the derived models can also be automatically translated into event based monitoring configurations. Hence a user who has to monitor the occurrence of an event for some aims, only needs to specify the event and activate the Model2Code transformation to instrument the event-based GLIMPSE monitor [5].

In the remaining part of this paper, we provide in Section II an overview of most related approaches, in Section III a brief overview of the PMM and the GLIMPSE configuration approach. In Section IV we describe in detail the proposed complex events specification language and its operators, comparing them with those of existing languages. In Section V, we show the models of two complex events and the concrete monitoring setup derived from one of them. Finally, conclusions (Section VI) complete the paper.

II. Related Work

Defining expressive complex event specification languages has been an active research topic for years [13], [6], [8]. In particular, the operators proposed into the PMM complex events specification language have been designed by taking into account the event composition operators addressed in GEM [13] and Drools Fusion [1]. GEM [13] is a declarative

and interpreted event monitoring language. It is rule-based (similar to other event-condition-action approaches) and provides a detection algorithm that can cope with communication delay. Similarly to GEM, our proposed language allows for specifying primitive and arbitrarily composite events. As in GEM, in PMM an event is a happening of interest but whereas in GEM an event occurs instantaneously at a specific time and a composite event may consist of a number of primitive events occurring at different times, in PPM complex event specification language it is possible to specify more accurate time constraints. Specifically, in PMM each simple event has a start and end timestamp and for complex events it is possible to specify some parameters for quantifying the maximum and minimum temporal distance between the time when the correlated event finishes and the current event starts.

In Drools Fusion [1] the events are special entities that represent an important change of state in the application domain; they have several characteristics, like being usually immutable, having strong temporal constraint and relationships. The set of operators used in Drools Fusion are temporal operators and allow for modeling and reasoning over temporal relationship between events. Drools supports the declaration and usage of point-in-time events and interval-based events.

The complex events specification language we present in this paper combines features of both languages and in addition presents new operators not included in the considered languages as showed in Section IV.

From one side, in GEM there are few basic composition operators that are used for defining the others but they are not enough for expressing all compositions operators of GLIMPSE monitor that includes a Drools Fusion [1] based engine. From the other side, Drools Fusion does not give the possibility to express operators such that *or*, *not* or *seq*. The idea in this paper is to combine the advantages of both languages in order to provide the developer with a more powerful, flexible and expressive specification language.

Other events specification languages address more specific application features. Among them, Snoop [6] follows an event-condition-action approach supporting temporal and composite events specification but it is especially developed for active databases.

Other proposals focus on formally defined approaches. A more recent formally defined specification language is TESLA [8] that has a simple syntax and a semantics based on a first order temporal logic. The authors of TESLA [8] also show as TESLA rules can be interpreted by a processing system, having an efficient event detection algorithm based on automata. TESLA considers incoming data items as notifications of events and defines how complex events can be defined from simpler ones. It provides content and temporal constraints, parameterizations, negations, sequences, aggregates, timers, and fully customizable policies for event selection and consumption but its clear and easy-to-use syntax allows for a limited number of different operators. Specifically, TESLA provides three event composition operators: *each-within*, *first-within*, and *last-within*. With respect to TESLA our work provides a more high-level and more specialized complex events specification language included into a comprehensive and flexible meta-model which defines monitoring goals (non-functional properties and metrics definitions).

An additional advantage of PMM events specification language is to be machine-processable and then it can be easily translated into rules of an existing open-source event processing engines that is Drools Fusion [1]. This engine can be fully embedded in existing Java architectures and provides efficient rule processing mechanisms.

The proposed complex events specification language is used to instruct the GLIMPSE monitor about non-functional properties to be checked at run-time.

Concerning monitoring systems, the literature is huge and rich of proposals of frameworks, languages, and architectures targeting functional and non-functional properties. Event-based monitoring is the most commonly used approach for observing the behavior of distributed systems. In particular, [16] presents an extended event-based middleware with complex event processing capabilities on distributed systems. Similar to GLIMPSE this work adopts a publish/subscribe infrastructure. Another monitoring architecture for distributed systems management is presented in [11]. This architecture employs a hierarchical and layered event filtering approach, specifically targeted at improving scalability and performance for large-scale distributed systems, minimizing the monitoring intrusiveness.

Other monitoring frameworks exist, that address mostly the monitoring of performance in the context of system management [3], [15], [2]. Among them, Ganglia [15] is especially dedicated for high-performance computing and is used in large clusters, focusing on scalability through a layered architecture.

Differently from the previous approaches focused on specific goals and constraints of the monitoring activity, GLIMPSE [5] is an implementation of a flexible and adaptable monitoring, developed with the goal of decoupling the event specification from the analysis mechanism. The GLIMPSE infrastructure is totally generic and can be easily applied to different contexts for supporting behavioral learning, performance and reliability assessment, security, and trust management. For this flexibility and the capability of decoupling the events specification from their detection and processing, GLIMPSE has been selected as monitoring infrastructure for runtime validations of PMM complex events models. Anyway, the events models specified with the language proposed in this paper can be used for instrumenting any monitoring system as long as a modelToCode Transformer transforms the models into the rule language of that monitoring engine.

III. Property-Driven Monitoring Configuration

In this section we recall some key concepts relative to monitor configuration (through the automatic translation of the models conforming to PMM into a concrete monitoring setup) and the general structure of the PMM.

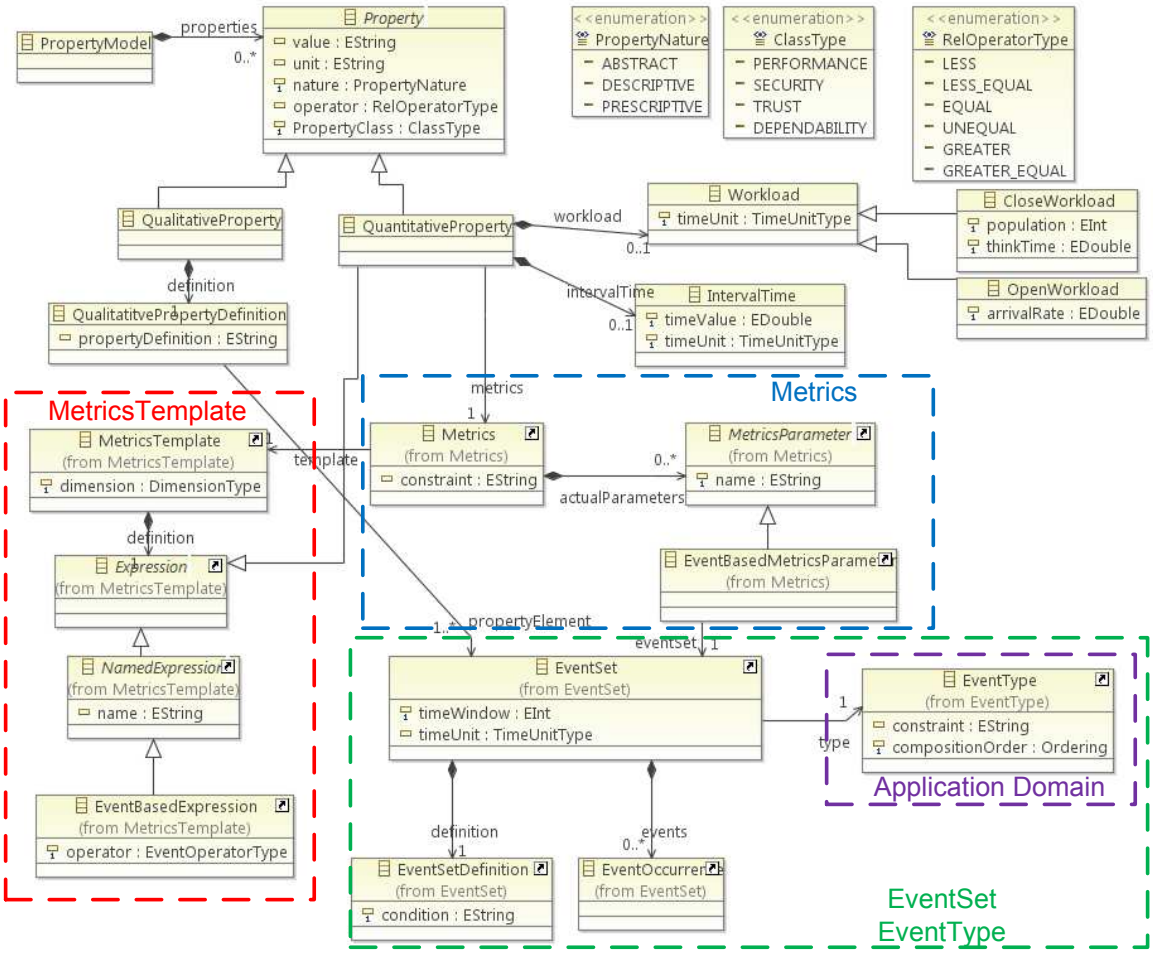

Fig. 1. Key Concepts of the Property Meta-Model

A. Monitoring Configuration

To instruct the monitor infrastructure about what raw data (events) to collect and how to infer whether or not a desired property is fulfilled is a time-consuming task, if a model-driven approach is not employed. This process would in fact need to be iterated each time the properties to be monitored change, it would also require a substantial human effort and specialized expertise if the high level description of the system properties to observe have to be translated into lower-level monitor configuration directives. Consequently, the outcome of such effort is very hard to generalize and to reuse, and, as a matter of fact, the resulting monitor configuration typically is only relevant in the specific situation at hand. To address such issues, we provided a model-based approach to automatically convert the PMM metrics, properties and event specifications into a concrete monitoring setup.

By focusing on the events, the editor provided along with PMM allows the software developer to specify a new complex event as a model that is conforming to PMM. A Model2Code Transformer translates this model into one or more rules according to Drools Fusion that is the current complex event processor embedded into our monitor infrastructure called GLIMPSE[1] [5].

The advantage of adopting this model-driven approach is that it allows the monitor to use any complex event processing engine as long as a Model2Code Transformer transforms the complex event model into the rule specification language of that processing engine. We presented a preliminary proposal of this approach in [4], [14]. Looking from the reverse perspective, we also have the advantage that the same PMM model could be used for any event-based monitor, by translating it into the monitor language.

B. PMM Concepts

The PMM [9] defines elements and types to specify prescriptive (required) and descriptive (owned) non-functional properties that the system under validation must provide or may expose, respectively. Figure 1 reports the key concepts of the whole meta-model and how they relate with each other. As the figure shows, a property can be qualitative

[1] http://labse.isti.cnr.it/tools/glimpse

(*QualitativeProperty*) or quantitative (*QuantitativeProperty*). In general, quantitative properties are related to performance or dependability, whereas qualitative properties are related to security (that is, a software system is secure or it is not). More precisely, *QualitativeProperty* refers to properties about the event occurrences of an *EventSet* that are observed and cannot be measured. They in general refer to the behavioral description of the system (e.g., deadlock freeness or liveness). Quantitative properties (*DESCRIPTIVE* or *PRESCRIPTIVE*) are measurable and have associated *Metrics*.

A *QuantitativeProperty* can have a *Workload* and/or an *IntervalTime*. The former is mandatory for *PERFORMANCE* properties, while the latter is mandatory for *DEPENDABILITY* ones.

As the figure shows, the other key concepts of PMM are the *MetricsTemplate* modeling a generic metric (e.g., latency as the duration of an event/operation), the concrete *Metrics* that refers to a *MetricsTemplate* and specifies the (actual) *metricParameter* for the (formal) *templateParameter* in the particular application domain the metric has been defined for. The specification of the concrete event in the metric is done via the concept the *EventType* the metric refers to. Such *EventType* refers to observable operations or events belonging to the ontology of the Application Domain for which the *Property* is relevant or it needs to be verified/guaranteed. An *EventSet* represents a set of event instances the refer to the same *EventType*.

PMM is implemented as an eCore model using Eclipse Modeling Framework (EMF) [10], [12] and it is provided with an associated editor realized as an Eclipse Plugin. This editor contains the information of the defined models and allows for creating new model instances of the *Property*, *Metrics*, *MetricsTemplate*, *EventType* and *EventSet* meta-models. The ultimate vision we want to achieve is that a software developer using PMM can either retrieve from the editor repository the pre-built specification of a simple or complex property or, if not present, can build new properties and metrics of interest, using the concepts in the meta-model. More details about PMM can be found on line[2] and in [9].

In the rest of this work we focus on the *EventType* meta-model, which is the original contribution of this work with respect to [9]. While in the previous released version of the meta-model this specification was in fact simply defined by means of a label or string, we have now formally specified the meta-model supporting complex event definition, as we detail in the following.

IV. PMM Complex Events Specification

In this section we detail the complex events specification language included into PMM. As depicted in Figure 2, the *EventTypeModel* of the PMM is composed by zero or more *EventType* elements, each one modeling the type of an event. The *EventType* models an observable system behavior that can be a simple event or operation representing the lowest observable system activity or a complex event that is a combination of simple and other composite events. An *EventType* has one or more parameters, one constraint and is composed by one or more *ComplexEvent*.

The *ComplexEvent* is a combination of simple and other composite events, combined by means of the operators defined in *OperatorType*. The required *compositionOrder* attribute represents the order of the events in the composition and can take one of the values listed in the *Ordering* enumeration. The *OperatorType* of *EventType* model can be one of the following operators:

- *After* operator: it involves two events and occurs when the current event happens after the correlated event. This operator uses two parameters to quantify the temporal distance between the time when the correlated event finishes and the current event starts: the former indicates the minimum distance while the latter indicates the maximum one. These parameters are called *minDistance* and *maxDistance*.
- *AfterT* operator: it involves one event and a time-period, it occurs when the event happens after the specified time-period. A parameter is defined corresponding to the time-period.
- *Before* operator: it involves two events and occurs when the current event happens before the correlated event. This operator uses two parameters to quantify the temporal distance between the time when the current event finishes and the correlated event starts, the former indicates the minimum distance while the latter indicates the maximum one. These parameters are called *minDistance* and *maxDistance*.
- *BeforeT* operator: it involves one event and a time-period, it occurs when the event is followed by a specified time-period. A parameter is defined corresponding to the time-period.
- *Coincides* operator: it involves two events and occurs when both happen at the same time, specifically the two events have the same start and end timestamps. This operator accepts one or two parameters, if only one parameter is defined, it represents the maximum distance between the corresponding timestamps while if two parameters are defined, the former represents the maximum distance between the start timestamps while the latter represents the maximum distance between the end timestamps. We implemented two versions of this operator: the former is implemented by the *Coincides_1p* operator and represents the behavior when only one parameter is defined (*maxDistanceTS*); the latter corresponds to *Coincides_2p* operator that represents the behavior using two parameters, these parameters are called *maxDistanceStartTS* and *maxDistanceEndTS*.
- *Concurrent* operator: it involves two events and occurs when both events happen irrespective of their order.
- *During* operator: it involves two events and occurs when

[2]http://labse.isti.cnr.it/tools/pmm

TABLE I
MAPPING WITH GEM AND DROOLS OPERATORS

PMM operator	GEM operator	Drools operator
A After B $(minDistance = x)$ $(maxDistance = y)$	(B+t); A $x \leq t \leq y$	A after B[x,y]
A AfterT 10	-	-
A Before B $(minDistance = x)$ $(maxDistance = y)$	(A+t); B $x \leq t \leq y$	A before B[x,y]
A BeforeT 10	A + 10	-
A Coincides_1p B $(maxDistanceTS = x)$	$(((A_s + t); B_s)I((B_s + t); A_s)))\&(((A_e + t); B_e))I ((B_e + t); A_e))t \leq x$	A coincides B[x]
A Coincides_2p B $(maxDistanceStartTS = x)$, $(maxDistanceEndTS = y)$	$(((A_s + t_1); B_s)I((B_s + t_1); A_s)))\&(((A_e + t_2); B_e))I ((B_e + t_2); A_e))t_1 \leq x; t_2 \leq y$	A coincides B[x,y]
A Concurrent B	$A\&B$	-
A During_2p B $(maxDistanceTS = x)$	$((B_s + t); A_s)\&((A_e + t); B_e)t \leq x$	A during B[x]
A During_2p B $(maxDistanceTS = x)$ $(minDistanceTS = y)$	$((B_s + t); A_s)\&((A_e + t); B_e)x \leq t \leq y$	A during B[x,y]
A During_4p B $(minDistanceStartTS = x)$ $(maxDistanceEndTS = y)$ $(minDistanceEndTS = u)$ $(maxDistanceStartTS = z)$	$((B_s + t_1); A_s)\&((A_e + t_2); B_e)x \leq t_1 \leq y; u \leq t_2 \leq z$	A during B[x,y,u,z]
A FinishedBy B $(maxDistanceEndTS = x)$	-	A finishedby B[x]
A Finishes B $(maxDistanceEndTS = x)$	$(B_s; A_s)\&(((A_e + t); B_e)I ((B_e + t); A_e))t \leq x$	A finishes B[x]
A FollowOut B,C	$\{A; B\}!C$	-
A Includes_2p B $(minDistanceTS = x)$	$B_e)t \leq x$	A includes B[x]
A Includes_2p B $(maxDistanceTS = x)$ $(minDistanceTS = y)$	$B_e)x \leq t \leq y$	A includes B[x,y]
A Includes_4p B $(minDistanceStartTS = x)$ $(minDistanceEndTS = y)$ $(maxDistanceEndTS = u)$ $(maxDistanceStartTS = z)$	$B_e)x \leq t_1 \leq y; u \leq t_2 \leq z$	A includes B[x,y,u,z]
A Meets B $(maxDistance = x)$	-	A meets B[x]
A MetBy B $(maxDistance = x)$	-	A metby B[x]
Not A	-	-
A Or B	AIB	-
A OverlappedBy_1p B $(maxDistance = x)$	-	A overlappedby B[x]
A OverlappedBy_2p B $(minDistance = x)$ $(maxDistance = y)$	-	A overlappedby B[x,y]
A Overlaps_1p B $(maxDistance = x)$	$(A_s; ((B_s + t); A_e)); B_s$ $t \leq x$	A overlaps B[x]
A Overlaps_2p B $(minDistance = x)$ $(maxDistance = y)$	$(A_s; ((B_s + t); A_e)); B_s$ $x \leq t \leq y$	A overlaps B[x,y]
SeqA$(min\_lenght = n)$	$A; A; A; A; \cdots$ (n times)	A after A after A after A $\cdots$ (n times)
SeqUniqueA$(min\_lenght = n)$	$A_1; A_2; A_3; \cdots A_n$	A_1 after A_2 after A_3 $\cdots$ after A_n
A StartedBy B $(maxDistanceStartTS = x)$	-	A startedby B[x]
A Starts B $(maxDistanceStartTS = x)$	$(((A_s + t); B_s)I((B_s + t); A_s))\&(A_s; B_s)t \leq x$	A starts B[x]

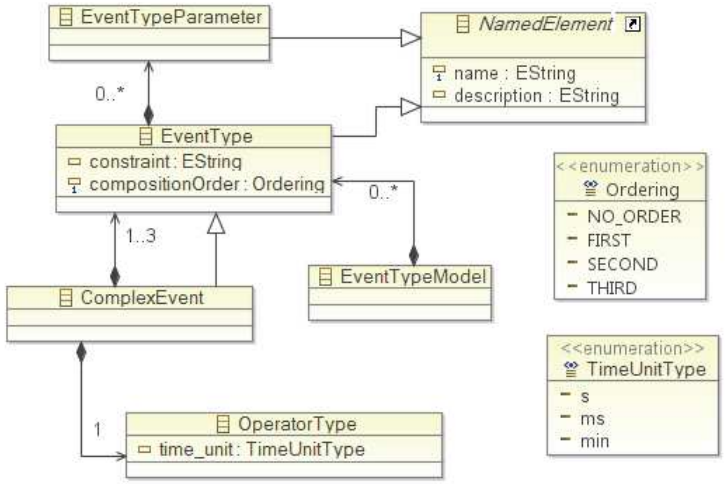

Fig. 2. EventType Meta-Model

the current event happens during the correlated event: specifically the current event starts after the correlated event and finishes before it. This operator accepts one, two or four parameters: if only one parameter is defined, it represents the maximum distance between the start timestamps of the two events and the maximum distance between the end timestamps; if two parameters are defined, the former represents the minimum distance between the timestamps while the latter represents the maximum distance between the timestamps; if four parameters are defined, the first two values represent the minimum and the maximum distance between the start timestamps while the other two values represents the minimum and the maximum distance between the end timestamps. We implemented two versions of this operator: the former is implemented by the *During_2p* operator that represents the behavior when one parameter (*maxDistanceTS*) or two parameters (*maxDistanceTS* and *minDistanceTS*) are defined; the latter is implemented by the *During_4p* operator that represents the behavior when four parameters (*minDistanceStartTS*, *maxDistanceStartTS*, *minDistanceEndTS* and *maxDistanceEndTS*) are defined.

- *FinishedBy* operator: it involves two events and occurs when the current event starts before the correlated event but both events end at the same time. This operator accepts one parameter (*maxDistanceEndTS*) that indicates the maximum distance between the end timestamps.
- *Finishes* operator: it involves two events and occurs when the current event starts after the correlated event but both events end at the same time. This operator accepts one parameter, called *maxDistanceEndTS*, that indicates the maximum distance between the end timestamps.
- *FollowOut* operator: it involves three events and occurs when the first event is followed by the second event and without the occurrence of the third event.
- *Includes* operator: it involves two events and occurs when the correlated event happens during the current event, specifically the correlated event starts after the current event and finishes before it. It is the symmetrical opposite of the *During* operator. This operator accepts one, two or four parameters: if only one parameter is defined, it represents the maximum distance between the start timestamps of the two events and the maximum distance between the end timestamps; if two parameters are defined, the former represents the minimum distance between the timestamps while the latter represents the maximum distance between the timestamps; if four parameters are defined, the first two values represent the minimum and the maximum distance between the start timestamps while the other two values represent the minimum and the maximum distance between the end timestamps.
  We implemented two versions of this operator: the former is implemented by the *Includes_2p* operator that represents the behavior when one parameter (*maxDistanceTS*) or two parameters (*maxDistanceTS* and *minDistanceTS*) are defined; the latter is implemented by the *Includes_4p* operator that represents the behavior when four parameters (*minDistanceStartTS*, *maxDistanceStartTS*, *minDistanceEndTS* and *maxDistanceEndTS*) are defined.
- *Meets* operator: it involves two events and occurs when the current event finishes at the same time when the correlated event starts. This operator accepts one parameter, called *maxDistance*, that indicates the maximum distance between the end timestamp of the current event and the start timestamp of the correlated event.
- *MetBy* operator: it involves two events and occurs when the current event starts at the same time when the correlated event finishes. This operator accepts one parameter, called *maxDistance*, that indicates the maximum distance between the end timestamp of the correlated event and the start timestamp of the current event.
- *Not* operator: it involves one event and occurs when the

specified event doesn't happen.

- *Or* operator: it involves two events and occurs when one of the two events happens.
- *OverlappedBy* operator: it involves two events and occurs when the correlated event happens before the current event and finishes before the current event but after the current event starts. This operator accepts one or two parameters, if only one parameter is defined, it represents the maximum distance between the start timestamp of the current event and the end timestamp of the correlated event. If two parameters are defined, the first one represents the minimum distance while the second one represents the maximum distance between the start timestamp of the current event and the end timestamp of the correlated event. We implemented two versions of this operator: the former is implemented by the *OverlappedBy_1p* operator that represents the behavior when only one parameter (*maxDistance*) is defined, the latter is implemented by the *OverlappedBy_2p* operator that represents the behavior when two parameters (*minDistance* and *maxDistance*) are defined.
- *Overlaps* operator: it involves two events and occurs when the current event start before the correlated event and finishes before it but after that the correlated event starts. This operator accepts one or two parameters: if only one parameter is defined it represents the maximum distance between the start timestamp of the correlated event and the end timestamp of the current event. If two parameters are defined, the former represents the minimum distance while the latter represents the maximum distance between the start timestamp of the correlated event and the end timestamp of the current event. We implemented two versions of this operator: the former is implemented by the *Overlaps_1p* operator and represents the behavior when only one parameter is defined (*maxDistance*); the latter corresponds to *Overlaps_2p* operator that represents the behavior using two parameters, these parameters are called *minDistance* and *maxDistance*.
- *Seq* operator: it involves one event and occurs when there is a sequence of occurrences of it. This operator accepts one parameter (*minLenght*) that indicates the minimum length of the sequence.
- *SeqUnique* operator: it is similar to the *Seq* operator. The additional feature is that the sequence captured by *SeqUnique* doesn't contain duplicate occurrences of the events.
- *StartedBy* operator: it involves two events and occurs when both events start at the same time and the correlated event finishes before the current one. This operator accepts one parameter (*maxDistanceStartTS*) that indicates the maximum distance between the start timestamps of the events.
- *Starts* operator: it involves two events and occurs when both events start at the same time and the current event finishes before the correlated event. This operator accepts one parameter (*maxDistanceStartTS*) that indicates the maximum distance between the start timestamps of the events.

The complex events specification language proposed in this paper combines features of two existing event specification languages that are GEM [13] and Drools Fusion [1] (see Section II) and in addition presents new features not included in the considered languages. In particular, we have defined operators that allow for modeling a temporal relationship (as those of Drools Fusion) and operators that allow for combining simple or complex events (as those of GEM), in addition we have identified situations of interest not covered by the operators of GEM and Drools Fusion that have been formalized through new operators. In Table I we present a mapping between the operators of our complex events specification language and the corresponding ones in GEM and Drools Fusion. In the table the symbol - means that there does not exist any correspondence with PMM event specification operator. Moreover, for some PMM operators (such as *SeqA*) there is not an equivalent GEM/Drools operator, but we are able to define a complex event in the target language by using its native operators (in case of *SeqA* we combine several time the *;* operator for GEM and *after* for Drools). Note that in the table, A_s and B_s represent the start timestamps of the events A and B respectively, similarly A_e and B_e represent the end timestamps of A and B.

V. Application Example

In this section we show the model of two complex events, namely *eAlert_eAck* and *contract*, and their usage in coverage and privacy properties in the Terrorist Alert Scenario[3], respectively. It depicts the critical situation that during the show the stadium control center spots one suspect terrorist moving around. The alarm is immediately sent to the Policemen, equipped with ad hoc handheld devices which are connected to the Police control center to receive commands and documents, for example a picture of a suspect terrorist. Unfortunately, the suspect is put on alert from the police movements and tries to escape, evacuating the stadium. The policeman that sees the suspect running away can dynamically seek assistance to capture him from civilians serving as private security guards in the zone of interest. To get help in following the moves of the escaping terrorist and capturing him, the policeman sends to the civilian guards an alert message in which a picture of the suspect is distributed. On their side, to perform their service, the guards that control a zone are connected in groups and are equipped with smart radio transmitters. The guards control center sends an alert message to all guards of the patrolling groups; the message reports the alert details. On correct receipt of the alert, each guard's device automatically sends an ack to the control center.

Section V-A reports on the *eAlert_eAck* event type, required by the coverage property, and the corresponding Drools rule

[3]The Terrorist Alert Scenario is one of the demonstration examples chosen in the CONNECT project [7].

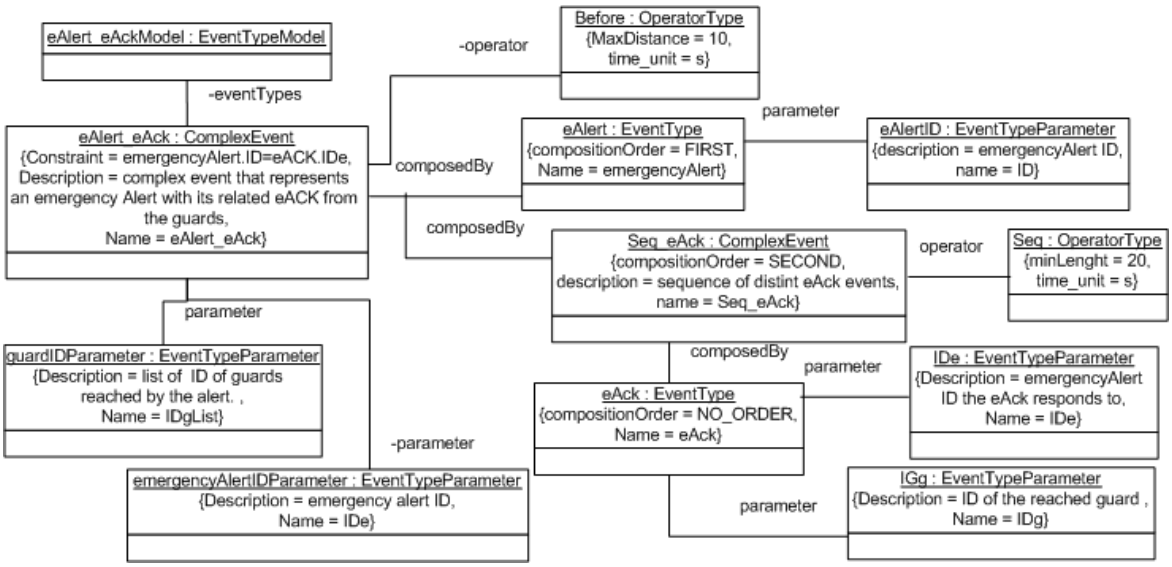

Fig. 3. Sequence of Ack for an Alert

generated by our Model2Code transformer. Section V-B deals with the *contract* event type required by the private security guards, through which we show the benefit of the combination of the GEM and Drools features and of the extension we introduce to the proposed language to make it more powerful w.r.t. the reference technology.

A. `eAlert_eAck` Event Type

The *eAlert_eAck* event type is an event to be monitored to measure the Delivery ratio, that is a dependability property for the Terrorist Alert Scenario. This property sets a minimum coverage measure that the system must satisfy. It will be measured by the percentage of guard devices that are reached by the alert message in a given time interval. This property depends on the event type represented by an *EmergencyAlert* message followed by a set of *eAck* coming back from the reached guards devices. We call such event type *eAlert_eAck*.

The *eAlert_eAck EventType*, shown in Figure 3, is a complex event representing the *EmergencyAlert* with its related *eAck* from the guards. The *Constraint* attribute defines the related condition that imposes that all the *eAck.IDe* must be equal to the *emergencyAlert.ID*. The *eAlert_eAck EventType* has a *Before* operator with *maxDistance* parameter equal to 10. This operator is applied to *eAlert* simple *EventType* and to *Seq_eAck* complex *EventType* representing respectively the former and the latter events to which the *Before* operator is applied. The *eAlert EventType* has the *eAlertID* parameter representing the *EmergencyAlert* ID the sequence refers to. The *Seq_eAck* is another *ComplexEvent* type with *Seq* operator (see Table I). It is composed by *eAck EventType*, with two parameters: *IDg* that is the ID of the reached guard and *IDe* that is the *EmergencyAlert* ID the *eAck* responds to. In this case the event *compositionOrder* is *NO_ORDER*. The *eAlert_eAck EventType* has two parameters: the *EmergencyAlert* ID (namely IDe) the sequence refers to, and the list of guards messages acknowledging the alert (namely, IDgList).

Listing 1 shows the Drool Fusion rule derived by the *eAlert_eAck* model applying the model driven monitoring configuration approach described in Section III. Specifically, this rule counts the number of *eAler_eAck* events that happen in a time window of 10 seconds and saves this information into a new generated event (called *counteAlert_eAck*).

```
1  declare Total_eAlert_eAckcaptured
2   @total: int
3  end
4
5  rule "Number of eAlert_eAck incomingEvents"
6  no-loop
7  salience 999
8  dialect "java"
9  when
10  $totaleAlert_eAck: Number();
11   from accumulate(
12  $event_eAlert_eAck: emergencyAlert( this before
13  $event_seq_eAck:
14  eAck( this after eAck after eAck after eAck after
        eAck after eAck after eAck after eAck after
       eAck after eAck after eAck after eAck after
       eAck after eAck after eAck after eAck after
       eAck after eAck after eAck after eAck )
15  )over window:time(10s) from entry-point "DEFAULT"
       , count($event_eAlert_eAck))
16 then
17  Total_eAlert_eAckcaptured counteAlert_eAck = new
       Total_eAlert_eAckcaptured();
18  counteAlert_eAck.setTotal($totaleAlert_eAck)
19  insert(counteAlert_eAck);
20  System.out.println( "Number of Incoming events: "
       + $totaleAlert_eAck);
21 end
```

Listing 1. Drools Code Generated by Model2Code Transformer for the *eAlerteAck* Complex Event

B. `Contract` Event Type

The *Contract* EventType is related to the private guards of the Terrorist Alert Scenario. Such security property requires

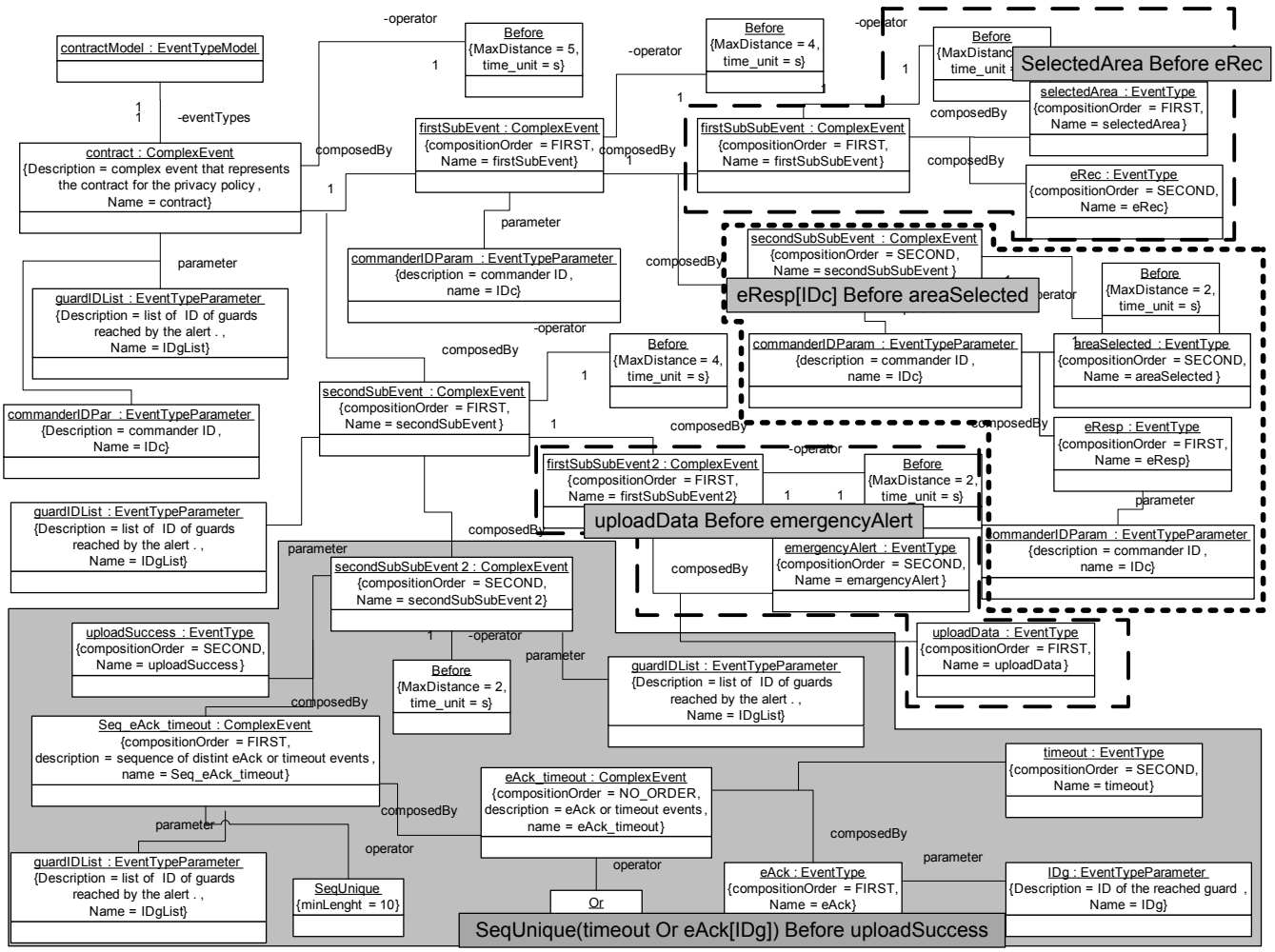

Fig. 4. The Contract Event Type

that *the photo can only be received by authorized devices*. This means that authorized devices: *i)* have a certain level of trust; *ii)* have to send an ACK; *iii)* expose a certificate for proving who and where they are.

More precisely, the private security guards definition sets that the *contract* among the parties must satisfy *sendAck* and *certification* EventType that are specification of the points *ii)* to *iii)* above[4]. This means that during system execution, when observing the communication among parties (modeled by the contract), the monitoring system must observe both *sendAck* and *certification* EventType.

Figure 4 shows the specification of the *contract* EventType whereas we here do not report *sendAck* and *certification* EventType since they have a specification similar to and simpler than *contract*, thus they do not add significant insights to the expressiveness of the proposed event specification language.

In a textual specification, that we here report to help the reader in reading the model in Figure 4, the *contract* is specified as follows:

```
((SelectedArea Before eRec) Before
(eResp[IDc] Before areaSelected)) Before
((uploadData Before emergencyAlert) Before
(SeqUnique(timeout Or eAck[IDg]) Before
uploadSuccess))
```

where IDg and IDc represent the guard and the commander ID, respectively, that are two different actors in the considered Terrorist Alert Scenario. Such IDs are parameters for the related events and are needed to identify specific event occurrences.

In Figure 4 we report for each portion of the event type the corresponding event specification written in the syntax above. In the figure, we shadow the submodel corresponding to the complex event `(SeqUnique(timeout Or eAck[IDc]) Before uploadSuccess)` to focus the attention of the reader on the most significant part of the event type specification w.r.t. the expressiveness of the proposed language. Indeed, to concisely and generally express such an event we use: *i)* event operators coming from both Drools [1]

[4]Point i) of the privacy property definition refers to a numerical value that is not an event to be monitored.

(i.e., the *Before* event operator) and GEM language [13] (i.e., the *Or* operator) showing the importance of the combination of the two reference event specification languages; and *ii)* the novel event operator *SeqUnique* we introduce to extend the expressiveness of the both reference languages. Without such combination and extension we were not able to express such complex event both in Drools and in GEM languages.

For what concerns the transformation of the *contract* event type into Drools rules, it is worth to note that at the moment the transformer is not able to perform such translation since, as reported in Table I, there is no correspondence of the *Or* operator in Drools Fusion language. As future work, we plan to fix this aspect by for example extending Drools and implementing the event operators that are still not covered by it.

VI. Conclusions and Future Work

Complex events definition is an important issue into non-functional properties validation. We proposed a machine processable specification language for complex events and compared it with other existing languages. The proposed language has been included into PMM and used for deriving an automated setup of an event-based monitoring infrastructure. As a proof of concept, we showed the application of the proposed approach to two complex events derived from the CONNECT project application scenarios.

As future work, taking into account other application scenarios, we plan to evaluate the limitations in terms of expressiveness of the proposed language and eventually define new operators. Moreover, we want to compare PMM with other specification languages such as TESLA [8] and TimeML [17] and at the same time we want to extend PMM to support other types of non-functional properties.

Acknowledgment

This work is partially supported by the EU-funded CONNECT project (FP7–231167) and by the EU-funded VISION ERC project (ERC-240555).

References

[1] Drools fusion: Complex event processor. http://www.jboss.org/drools/drools-fusion.html.

[2] Java Enterprise System Monitoring Framework. http://download.oracle.com/docs/cd/E19462-01/819-4669/geleg/index.html.

[3] W. Barth. *Nagios. System and Network Monitoring*. No Starch Press, u.s. ed edition, 2006.

[4] A. Bertolino, A. Calabrò, F. Lonetti, A. Di Marco, and A. Sabetta. Towards a model-driven infrastructure for runtime monitoring. In *Proceedings of SERENE*, pages 130–144, 2011.

[5] A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. Glimpse: a generic and flexible monitoring infrastructure. In *Proceedings of EWDC*, pages 73–78, 2011.

[6] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[7] CONNECT Consortium. Deliverable 6.1-Experiment scenarios, prototypes and report. http://connect-forever.eu/, 2011.

[8] G. Cugola and A. Margara. TESLA: a formally defined event specification language. In *Proceedings of DEBS*, pages 50–61, 2010.

[9] A. Di Marco, C. Pompilio, A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. Yet another meta-model to specify non-functional properties. In *Proceedings of QASBA*, pages 9–16, 2011.

[10] Eclipse Platform, Eclipse Modeling Project. http://www.eclipse.org/modeling/.

[11] E. Al-Shaer Hussein, H. Abdel-wahab, and K. Maly. HiFi: A New Monitoring Architecture for Distributed Systems Management. In *Proceedings of ICDCS*, pages 171–178, 1999.

[12] L. Vogel. Eclipse Modeling Framework Tutorial, http://www.vogella.de/articles/EclipseEMF/article.html#emfoverview, January 2011.

[13] M. Mansouri-Samani and M. Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[14] A. Di Marco, F. Lonetti, and G. De Angelis. Property-driven software engineering approach. In *Proceedings of ICST*, pages 966–967, 2012.

[15] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 840, 2004.

[16] P.R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *Network, IEEE*, 18(1):44 – 55, jan. 2004.

[17] J. Pustejovsky, J. Castao, R. Ingria, R. Saur, R. Gaizauskas, A. Setzer, and G. Katz. TimeML: Robust specification of event and temporal expressions in text. In *Proceedings of IWCS-5*, 2003.