Yet Another Meta-Model to specify Non-Functional Properties

Antinisca Di Marco, Claudio Pompilio
University of L’Aquila
Coppito (AQ), Italy
antinisca.dimarco@univaq.it, claudiopompilio@gmail.com

Antonia Bertolino, Antonello Calabrò, Francesca Lonetti
Istituto di Scienza e Tecnologie dell’Informazione “A. Faedo”, CNR
Pisa, Italy
firstname.secondname@isti.cnr.it

Antonino Sabetta†
SAP Research
Sophia Antipolis, France
antonino.sabetta@sap.com

ABSTRACT
In service-oriented systems non-functional properties become very important to support run-time service discovery and composition. Software engineers should take care of them for guaranteeing the service quality in all the software life-cycle phases, from requirements specification to design, to system deployment and execution monitoring. This wide scope and the criticality of non-functional properties demand that they are expressed in a language which is intuitive and easy to use for the service quality specification, and at the same time is machine-processable to be automatically handled at run-time. In this paper we present a Property Meta-Model that aims to reach these two main objectives and show as a proof of concept its use for the modeling of two different properties.

Categories and Subject Descriptors
D.2 [Software Engineering]: Requirements/Specifications—languages; D.2.8 [Software Engineering]: Metrics—performance measures; D.2.9 [Software Engineering]: Management—Software quality assurance (SQA)

General Terms
Languages, Measurement

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

1. INTRODUCTION
In service-oriented systems, the capability of handling non-functional properties becomes more and more important for run-time specification by the requesters of the expected quality of service delivery and for dynamic selection among alternative services in a composition. Therefore, a machine-processable specification of the non-functional properties that services must guarantee becomes an essential asset for the run-time integration of heterogeneous software services. Such specification can be referred to for the continuous monitoring of the service operation.

In this respect an effective and popular approach nowadays is the model-driven paradigm, by which we can deal separately with the concerns of structuring the specification of properties and the domain-specific concepts belonging to one application. The abstract property structure is defined into a generic meta-model; this is then instantiated into a concrete property model by considering the elements belonging to a specific service application. Goal of this paper is to present a meta-model to specify non-functional properties for the quality assurance of service-based applications.

Model-driven specification of non-functional properties is not a new idea, on the contrary (as we report later in the Related work Section) it is now a well established principle, which has been developed in several contexts.

So, why yet another meta-model? Our contribution stays in the effort (or ambition) to conceive a comprehensive machine-processable meta-model for non-functional properties that spans over dependability, performance, security and other complex properties (such as trust) that can be defined upon others properties and need to be guaranteed in service based applications. From our overview of the literature, existing meta-models generally address only a subset of the above properties, or embed service application specific concepts or do not support transformational approaches.

More in detail, our Property Meta-Model (PMM) has been devised by taking into account the following desirable features:

• comprehensiveness: by using PMM, it is possible to
specify performance as well as dependability, security, and even (service) trust properties.

- flexibility: the definition of a metric must be independent on the service-application domain it is used for (e.g., latency can be modeled as the difference of two time-stamps, this definition is always valid whatever the target system is); this permits to define once the generic metrics and then use it for all future aims;

- application-independence: PMM allows for the specification of simple and complex events associated to a template which can be tailored to describe the intended semantics when defining an application-specific metrics;

- support for many transformations: on top of the PMM meta-model we can implement model-to-model transformations for several purposes. An example on how to use PMM models to instrument a generic run-time monitoring infrastructure to monitor non-functional properties is in [3].

The defined PMM is implemented as an eCore model and it is provided with an associated editor realized as an Eclipse Plugin. 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 service property or, if not present, can build new properties and metrics of interest, using the concepts in the meta-model.

Moreover, using PMM it is possible to specify generic properties and metrics of a system and not only specific service properties.

In the remaining part of this paper, after providing an overview of most related approaches (Section 2), we describe in detail the Property Meta-Model (PMM) we defined to specify service properties (Section 3). Then in Section 4, we show how to model some relevant properties for an application scenario. Finally, conclusions and future work (Section 5) complete the paper.

2. RELATED WORK

Several works addressed meta-modeling focusing on domain-specific metrics or dependability properties [15, 9] among which it is worth mentioning:

- Quality of Service Modeling Language (QML) [6] for describing QoS specifications for software components in distributed object systems. It is an extension of UML, allowing a fine-grained specification level of attributes and operations and a dynamic and runtime check of QoS components requirements and dependencies;

- UML MARTE profile [14] that provides a common way for modeling hardware and software aspects of a real time embedded system. It provides facilities to annotate models with information required to perform quantitative predictions and performance analysis;

- a model-driven performance measurement and assessment approach [4] that provides a meta-model for the specification of performance metrics and the possibility of specifying measurement points into the model. It allows the automatic instrumentation and software code generation with integrated code for performance data collection, storage and metrics computation.

- a Quality-Value-Dependency-Priority (QVDP) model [10] that integrates different models onto a quality model for service-oriented systems and it is used for extending the UML QoS meta-model [8].

Differently from the previous approaches, PMM is a more general and complete framework, that allows the specification of different types of properties, such as, among the others, performance, security, dependability and complex properties (based on other descriptive properties) and, when applicable, of relative metrics.

A new research direction focuses on ontologies for modeling QoS with rich semantic information. In particular, a semantic QoS model for dynamic service environments is presented in [11]. It makes use of Web Service Quality Model (WSQM) standard to define QoS at the service level and addresses four ontologies specifying respectively: the core QoS concepts, the environment and underlying network and hardware infrastructure QoS properties, the application server and user-level QoS properties. The authors do not provide a language to specify QoS but separate and reusable ontologies to be used for specifying a language. With respect to this approach, PMM provides a specification language for non-functional properties.

The main concept underlying our proposal, i.e., specifying metrics as instances of a metric specification meta-model, is common to the work of Monperrus et al. [13], in which a generative model driven definition of software metrics is proposed. This work concerns the definition of a domain-independent metrics meta-model, allowing modelers to automatically add measurement capabilities to a domain specific modeling language used during the different phases of a model-driven development process. Taking inspiration from the work of Monperrus et al., PMM separates the property and more specifically metrics definition from the application domain. Instead, differently from [13], the PMM meta-model addresses specifically non-functional property and metrics by introducing additional concepts concerning the qualitative and quantitative properties definition and the events modeling that is used to specify the semantics of the metrics/properties for a specific application.

3. PROPERTY META-MODEL

In this section, we present the structure of the meta-model we defined for specifying observable properties of a service-based system. The key concepts of this meta-model are: Property, MetricsTemplate, Metrics, EventSet, and EventType. As in [13], we separate the property definition from the application domain and its specific ontology. The ontology is linked to the Property meta-model via the EventType entity that models a generic event type where the terms of the application-domain ontology will be used.

In the following we provide the description of the proposed meta-model by giving details on the Property meta-model (in Section 3.1); Metrics and MetricsTemplate meta-model (in Section 3.2); and finally on the EventSet and EventType definition meta-model (in Section 3.3). The editor associated to PMM contains the information of the defined models and allows to create new model instances of the Property, Metrics, MetricsTemplate, EventType and EventSet meta-models1.

1A release of the Property Meta-Model and associated editor is available at http://labse.isti.cnr.it/tools/cpmm
3.1 Property definition

Figure 1 reports the portion of the meta-model describing a Property. A PropertyModel is composed by zero (modeling the empty model) or more properties.

A Property is a NamedElement having two required/-mandatory attributes and three optional ones. The required attributes are: nature and PropertyClass. The nature attribute refers to the nature of the property that can be ABSTRACT, DESCRIPTIVE, or PRESCRIPTIVE, the PropertyClass can have the following values: PERFORMANCE, SECURITY, TRUST, and DEPENDABILITY.

An ABSTRACT property indicates a generic property that does not specify a required or guaranteed value for an observable or measurable feature of a system. A DESCRIPTIVE property represents a guaranteed/owned property of the service-based system while a PRESCRIPTIVE one indicates a system requirement. In both cases, the property is defined taking into account a relational operator with a specified value. The optional attributes of Property are value, unit and operator. These attributes are not specified in case of an ABSTRACT property because, as described above, an ABSTRACT property does not specify a relation with a specific value (as it is required by the constraint in the upper note at the right-hand side of Figure 1). They are specified only for the DESCRIPTIVE and PRESCRIPTIVE properties. The value attribute indicates a value associated to the property and the unit attribute indicates its unit of measure whereas the operator attribute models a relational operator.

A property can be qualitative (QualitativeProperty) or quantitative (QuantitativeProperty). The former models properties about the event occurrences of an EventSet that are observed and can not be measured. They in general refer to the behavioral description of the system (e.g., deadlock freeness or liveness). The quantitative properties (DESCRIPTIVE or PRESCRIPTIVE) are measurable and they have an associated Metrics. The QuantitativeProperty can have a Workload or an IntervalTime. The former is mandatory for PERFORMANCE property while the latter for DEPENDABILITY one. The Workload can be open or close and it has the time-Unit attribute specified according to one of the time units listed in the TimeUnitType enumeration. The IntervalTime has the timeUnit and the timeValue attributes needed to specify the interval of time it represents. To clarify the above concepts we report below some Property examples:

- **[Property1:]** Ability to provide a service according to given time requirements. This is an ABSTRACT property since it does not deal with a claimed or guaranteed specified time feature. This property is also a QuantitativeProperty of the PERFORMANCE class because it is about a measurable performance dimension (time).

- **[Property2:]** The service $S$ in average responds in $3$ ms in executing the $e_1$ operation when it is subject to an open workload with arrivalRate of $10$ $e_2$ operations per time unit. This is a DESCRIPTIVE property asserting that the service $S$ guarantees in average a time response having a value of $3$ ms in executing the $e_1$ operation, with a workload of $10$ $e_2$ concurrent operations. As Property1, this one is also a QuantitativeProperty having a PERFORMANCE class because it is about a measurable performance dimension (time). In this case, the value attribute is equal to $3$, the unit measure is ms and the specified operator is EQUAL. The associated OpenWorkload is characterized by the arrivalRate equal to $10$ while intervalTime is not specified.

- **[Property3:]** The service $S$ in average must respond in $3$ ms in executing the $e_1$ operation when it is subject to an open workload with arrivalRate of $10$ $e_2$ operations per time unit. This is a PRESCRIPTIVE property because it specifies a required time response. As Property1 and Property2, this is also a QuantitativeProperty having a PERFORMANCE class. The value attribute, the unit of measure, the specified operator and the workload are equal to those of Property2.

In all the above examples, $S$, $e_1$ and $e_2$ are application-
dependent and their semantics can be specified once the application domain has been fixed.

3.2 Metrics and MetricsTemplate definition

In the proposed meta-model, we make a distinction between a generic metric (represented by a MetricsTemplate) and the concrete metric (i.e., the Metrics concept) instantiated to a specific application domain, represented by the domain of the software service the PMM is used for. Moreover, the MetricsTemplate is defined upon a generic set of actions/events/operations, that is not coupled with a particular application domain. This general part of the definition is specialized by the metric that instantiates those general concepts (templateParameters) with application-based actions/events/operations (metricsParameters). Hence a metric (for example the response time) can refer to one or more (hopefully equivalent) specifications represented by different MetricsTemplates. The response time of an operation (E), for example, can be expressed as the duration of that operation or the difference between the timestamps of the ending (E1) and starting (E2) actions of that operation. E, E1 and E2 are templateParameters. Figure 2 reports the meta-model portion describing the MetricsTemplate concept. A MetricsTemplateModel defines one or more MetricsTemplates, each having a dimension indicating the type of the value defined by the metrics template (e.g., a TIMEd value, a PERCENTAGE, and so on). A MetricsTemplate is a NamedElement containing the Expression describing the mathematical definition of the template. The Expression could be:

- a Constant which has a value attribute indicating the specific value the constant refers to.
- a MathExpression, this meta-class allows the definition of a complex mathematical expression by nesting one or more operands that are in turn other Expression.

The MathExpression has an attribute, operator, that specifies the MathOperatorType (listed in the MathOperatorType enumeration). We define some operators that are applied to single occurrences of simple or complex events (for example DURATION refers to a complex event and TIMESTAMP to a simple one), and other operators (such as CARDINALITY) that are applied to the whole set of event occurrences observed in a given instant of time.

- a QuantitativeProperty, already described, to be used as operand of an expression, it must have a DESCRIPTIVE nature.
- a NamedExpression. This element can represent: a canonical Variable, an ActionBasedExpression, representing a simple action or a sequence of actions that, whenever has been executed, reports a value, an EventBasedExpression, that represents expressions based on events or observational behavior.

The MetricsTemplate finally contains zero or more TemplateParameters. These template parameters are EString keys exposed by the template and linked to NamedExpression by mean of the EStringToNamedExpressionObjectMap concept.

Figure 3 reports the meta-model portion describing the Metrics concept. A MetricsModel defines zero or more Metrics. A metric is a NamedElement that refers to a MetricsTemplate, actualizes the templateParameters by means of the MetricsParameters. The EventBasedMetricsParameter is a MetricsParameter that actualizes the EventBasedExpression based templateParameters. To this goal, it refers to the EventSet describing the application based event definition and the relative occurrences.

The VariableMetricsParameters instead is a MetricsParameter actualizing the Variable templateParameters indicating the corresponding value.
Finally, Constraints defines some conditions on the EventSet involved in the Metrics. Such constraints in general allow to make a correspondence among event occurrences belonging to different types of EventSet.

3.3 EventType and EventSet definition

The EventSetModel (Figure 4) has zero or more EventSet. An EventSet represents a set of event instances that refer to an EventType. The property of an EventSet identifies all observable events that have to be included in the EventSet. An EventSet has zero or more EventOccurrence representing the observable events that the EventSet contains. An EventOccurrence refers to an Interval that represents the time range in which the observable event occurs. Each Interval has two associated Timestamps indicating its starting and ending date respectively. These Timestamps are equal in case of atomic/instantaneous event occurrences.

The EventType models an observable service behavior that can be a primitive/simple event or a composite/complex one. The ComplexEvent is a combination of primitive and other composite events 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.

For the sake of clarity, we present in Figure 5 only a scratch of the EventType model. The OperatorType of this model can be one of the operators described in Table 1, where e1 and e2 represent the current and correlated events respectively and e3 is a generic event involved in some operators. The proposed operators have been designed by taking into account the most common event composition operators addressed in [12, 1]. We refer to [2] for a more detailed description of them.

4. TERRORIST ALERT SCENARIO

The definition of PMM has been inspired by the FP7 “ICT forever yours” European Project CONNECT but has certainly wider application. The ambitious goal of the project is to have functioning systems within a dynamically evolving context, which is to be achieved by synthesizing on-the-fly the connectors through which heterogenous Networked Systems communicate. In this section we first outline the Terrorist Alert Scenario which is one of the demonstration examples chosen in CONNECT. Then, we show how to model using the PMM two different properties required in the system for that scenario: a latency property and a coverage one.

2http://connect-forever.eu
### Terrorist Alert scenario.

The CONNECT Terrorist Alert scenario [5], depicts the critical situation that during a 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, evading 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 some civilian guards, on service around the stadium, an alert message in which one picture of the suspect is distributed. The guards control center sends an EmergencyAlert 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. All actors involved in this scenario can be realized as components of a service-based system.

### Latency Property for the Terrorist Alert Scenario.

We show how to model the following required latency property: *average time needed by the CONNECTed system to reach k% guard devices must be at maximum equal to 10 seconds when in the system there are 10 alerts*. For “time needed by the CONNECTed system to reach a set percentage of guard devices” we mean the average latency experienced in the system in between the EmergencyAlert message and the reception of a percentage of eAck coming back from the reached guards’ devices. The model for this PRESCRIPTIVE property is shown in Figure 6. The property is a PERFORMANCE requirement specifying that the associated metrics (called LatencyReachingGuard) is LESS_EQUAL
to 10 s when the system has a workload of 10 alerts.

The LatencyReachingGuard metric actualizes the corresponding Average Latency Metrics Template (see Figure 7) by linking to the TemplateParameters the corresponding EventSets. Referring to the Terrorist Alert scenario, the LatencyMetricsTemplate is modeled as the DURATION of a complex EventType, hence it has only one TemplateParameter instantiated in the LatencyReachingGuard metric by the e EventSet. We recall that the template is generic and can be used in other scenarios, this shows the flexibility of the proposed meta-model. The e EventSet refers to eAckEventType. The eAckEventType, shown in Figure 8, 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.ID must be equal to the emergencyAlert.ID. eAckEventType has a Before operator with maxDistance parameter equal to 10. This operator is applied to eAck simple EventType and to Seq.eAck complex EventType representing respectively the former and the latter events to which the Before operator is applied. The eAck EventType has the eAckID parameter representing the EmergencyAlert ID the sequence refers to. The Seq.eAck is another ComplexEventType type with SeqUnique operator (see Table 1). It is composed by eAckEventType 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 NOORDER. The eAckEventType has two parameters: the EmergencyAlert ID (namely IDe) the sequence refers to, and the list of guards messages acknowledging the alert (namely, IDgList).

Coverage Property for the Terrorist Alert Scenario.

We show how to model the following required coverage property: the average percentage of guard devices that are reached in 10 seconds by the alert message must be greater than 70%. This means that, after 10 seconds from the EmergencyAlert, at least 70% of guard devices reply with an eAck. This property is a DEPENDABILITY requirement specifying that the associated metrics is GREATER than 70% after an IntervalTime of 10 seconds. The average coverage represents a PERCENTAGE measure defined as average of the division among the CARDINALITY of two sets of instances of two types of events, named x and y. Finally, the template exposes two templateParameters, e2 linked to x, and e3 linked to y (see Figure 9).

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented the PPM we devised to express service properties in a machine-processable way. The properties that can be specified are of three natures: abstract, prescriptive and descriptive. As proof of concept we have shown how the PPM can be used to instantiate two properties for the Terrorist Alert scenario.

There are various directions for future work. First, since the models that conform to such a meta-model are machine-processable, we plan to define automated procedures (in form of Model-To-Text transformations) that, from such models, instrument the monitoring for run-time verification of important service properties and produce suitable input for the analysis tools that should validate service properties.

Moreover, some meta-model parts need to be refined to address trust. In particular, i) the QualitativePropertyDefini-