Object Oriented Notation for Modelling Quantitative Aspects

Huszerl G., Dept. of Measurement and Information Systems, *Budapest Univ. of Technology and Econ.*, Hungary
K. Kosmidis, Dept. of Computer Science 3, *Friedrich-Alexander University Erlangen-Nuremberg*, Germany

Abstract

Nowadays formal methods and analysis techniques in design and modelling of modern computer controlled systems become more and more important. To provide easy-to-use tools for ensuring the overview of complex systems, multi-aspect modelling languages are specified (e.g. the Unified Modeling Language - UML). While focusing on best capturing the complex functionality, these languages neglect non-functional aspects such as quality of service - performance and dependability. However, during the modelling and design process, the specification of functional requirements is often insufficient. To deal with performance and dependability of systems the modelling languages should have an integral part of the notation for describing the quantitative properties of model elements. In the case of UML there are some ongoing research activities to extend it for dealing with such kind of data for real-time and high-assurance systems, but the current published standard proposals have their tight limitations. We have specified a language extending the UML to support stochastic modelling, performance and dependability analysis and modelling of hardware and software systems.

1 Introduction

As the importance of formal methods and analysis techniques in design and modelling of modern computer controlled systems increased during the last years, a wide variety of formalisms, languages and analysis techniques are offered to the designer. From the view of "design re-use" and tool support, standardised design languages are preferred. The Unified Modeling Language (UML) [1] provides a graphical notation (standardised by the Object Management Group [2]) for visualising, specifying, constructing and documenting the artefacts of complex distributed systems ranging from embedded systems to business applications. UML is supported by a wide variety of well-established tools and environments, offering services for specification, design refinement and automatic code generation. In the recent years, several methods were elaborated to enable also the formal analysis of UML based designs. Among others, problems of system-level dependability modelling, formal verification and performance analysis of UML (subset) models were solved [3].

Our work is focused on the quantitative dependability analysis of the UML behavioural models of embedded systems. The dynamic behaviour of the system is given in UML – among other diagrams – by statechart diagrams [2], an object-oriented mutation of classical Harel statecharts [4]. They describe the internal behaviour of components (objects, hardware nodes etc.) as well as their reactions to external events. The detailed description of the behaviour by statecharts enables dependability analysis, if the model is extended with explicit categorisation of failure states/events and probabilistic information.

While focusing on best capturing the complex functionality, the UML standard neglects non-functional aspects such as quality of service – performance and dependability. However, to deal with performance and dependability of systems the modelling languages should have an integral part of the notation for the description of quantitative properties of model elements.

To tailor the UML to particular application domains or to particular platforms the concept of profiles is provided. Currently there are profiles such as for Software Development Processes and Business Modelling, while other profiles are in progress.[1]

The next subsection describes the background, which has triggered this work. The second section introduces two specifications prepared independently in recent years, while the third section outlines some basic questions of the modelling for performance and dependability analysis. A notation for performance-related QoS characteristics is described in the fourth section, which will be completed by a notation for associating these characteristics with UML model elements in the next one. The last section concludes the work.

[1] In order to guard against confusion of terminology the name "profile" will be avoided in this context. Wherever in the next sections the word "profile" is mentioned, it is used in the QML context, and means a QML-profile, which is introduced in section 2.2.

1.1 Our analysis approach

Some of the ideas in this article are implicated by preceding research [3][5], which triggered the specification of this notation. While constructing our general framework for providing formal analysis techniques for complex safety critical systems, we proceed as follows (Fig. 1.):

1. The semi-formal system specification is described in UML.
2. The UML model is transformed to different formalisms such as to a special extension of finite automata (for qualitative analysis), to a special extension of Petri nets (for quantitative analysis) etc.
3. The generated mathematical models are analysed by existing analysis tools.
4. The results of the analysis are back-annotated in the UML model automatically.
5. The system modeller may edit his model and rerun the transformation–analysis–back-annotation cycle as many times as necessary before implementation, without deeper knowledge about the used formal methods and its tools.

Specially in quantitative analysis the modeller has to enrich the semi-formal model by quantitative data. These data can be derived from the non-functional requirements of the system, and then been specified and validated against the system model before the system is deployed. For this purpose the formal specification of the non-functional requirements is necessary.
If the requirements must be specified by a mathematical notation, even a modeller with strong mathematical background can have problems to specify them properly. Thus a requirement language that provides the required formalism to specify non-functional requirements and is intuitively enough to be applied is defined. As a user-front-end for the specification of non-functional requirements the language SQIRL (Stochastic Quantitative Requirement Language) was developed, that consists of English sentence fragments [6]. The fragments can be combined to build up sentences that describe requirements. They are characterised by a good readability. This is important since even people without mathematical background can understand SQIRL-Requirements. On the other hand they can be used for documentation purpose.
After describing the non-functional requirements by using SQIRL, the modeller has to specify them in context of the system model (Fig. 2). Since UML is not providing any appropriate notation we have specified a language extending UML, based on the following general requirements:

- The language has to be extensible to describe a wide range of quantitative (performance and dependability) data in modelling.

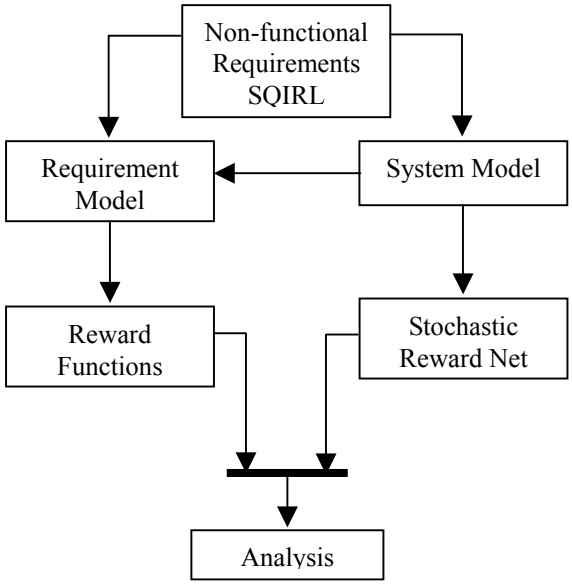

Fig. 2 Analysis of non-functional requirements on a model

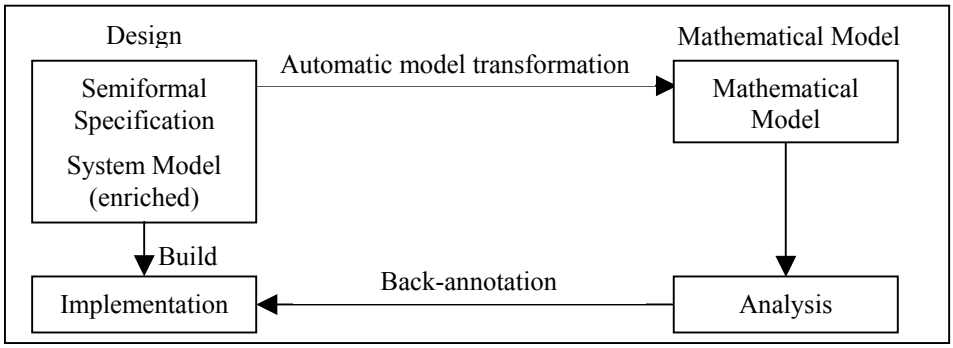

Fig. 1 Automated model analysis

- The language has to suit the UML, extending its descriptive power. It has to support the associations and hierarchy within the UML models.
- The language has to support fast input and easy linking of large amounts of data with a common value to individual model elements.
- The language has to support fast re-parameterisation of models.
- The language has to be able to represent queries for the analysis, this way aggregating the input and output data of it.

2 Referenced works

In this section two proposals are introduced which will be referred to in this paper. The first one is a proposal in progress at the Object Management Group (Status: The Technology Adoption vote has completed on November 9, 2001), which includes an approach and the UML extensions required to perform basic performance analysis of UML models. The second one is a general Quality-of-Service (QoS) specification language, which can be used to capture QoS properties as part of object-oriented designs.

2.1 The OMG proposal

The proposal "UML Profile for Schedulability, Performance, and Time, revised submission (OMG ad/2001-06-14)" [7] defines standard paradigms of use for modelling of time-, schedulability- and performance-related aspects of real-time systems that will:

- enable the construction of models that could be used to make quantitative predictions regarding these characteristics
- facilitate communication of design intent between developers in a standardised way
- enable inter-operability between various analysis and design tools.

The proposal contains among other things a specification of the general resource modelling, a generic model for representing time and time-related mechanisms, a general model of concurrency, and an approach to perform basic schedulability analysis. The 8th chapter of the proposal describes the approach and UML extensions required to perform basic performance analysis of UML models. In this paper we will show our approach on this concept.
However the authors of the proposal take cognisance of the importance of the ability to model resources and their performance related characteristics, they focus more on binding them to the model elements than on describing these characteristics. The proposed tagged-value-based annotation increases visual clutter, and it can be inconvenient for describing complex data structures (often necessary for presenting quantitative properties) and for working with a large number of identical data. One possible reason of this deficiency can be that the original "Request for Proposals" of the OMG has asked for proposals that are not changing the UML metamodel.
This proposal is referred to in our work as "the OMG proposal", notwithstanding that it is a joint submission by ARTISAN Software Tools, Inc, I-Logix Inc., Rational Software Corp., Telelogic AB, TimeSys – Corporation, Tri-Pacific Software Inc., and it is only submitted to the OMG for evaluation.

2.2 The Qos Modeling Language (QML)

Frølund and Koistinen [8] have proposed a general Quality-of-Service (QoS) specification language, which they call QML and which can be used to capture QoS properties as part of object-oriented designs. QML is designed to integrate object-oriented features, such as interfaces, classes, and inheritance. In particular, it allows specification of QoS properties through refinement of existing QoS specifications. Although we exemplify the use of QML to specify QoS properties within the categories of performance, QML can be used for specification within any QoS category - QoS categories are user-defined types in QML.
QML has three main abstraction mechanisms for the QoS specification: contract type, contract, and profile. QML allows us to define contract types that represent specific QoS aspects, such as performance or reliability. A contract type defines the dimensions that can be used to characterize a particular QoS aspect. A dimension has a domain of values that may be ordered. There are three kinds of domains: set domains, enumerated domains, and numeric domains. A contract is an instance of a contract type and represents a particular QoS specification.
Finally, QML profiles associate contracts with interfaces, operations, operation arguments, and operation results. For more details on QML we refer to the original papers with further informal [8][9] and formal [10] descriptions of the language. However, the language is human readable to such an extent, that the example code segments in this paper can be understood on the whole without further knowledge.

3 Quantitative analysis

While the qualitative analysis deals with the functional aspects of a system, other aspects such as performance, dependability, schedulability, timeliness

and others are the subjects of the quantitative analysis. (This work mainly focuses on performance and dependability, without losing sight of the other quantitative aspects.) The ability to predict these numeric characteristics based on the analysis of object-oriented models – notably including models that are constructed prior to implementation – is a fundamental objective of current research. Accurate and trustworthy predictions, while relying on mathematically derived results stemming from accurate models, invariably involve formal quantitative analyses. Rather than inventing new analysis techniques, the intention is to be able to annotate a UML model in such a way that various existing analysis tools will be able to take advantage of the provided features.

Our goal is to automatically derive mathematical models from the UML model of the target design under evaluation, and to hide the details of model analysis (including the specifics of its internal algorithms and data representation) from the modeller. It allows him to utilise the capabilities of the existing model analysis tools without being overwhelmed by the details of the mathematics and the tool.

To protect the modeller from the specifics of individual tools, all information entered and viewed by the modeller is included part of the model. To support the inclusion of quantitative aspects of the system under evaluation it is necessary to extend the UML.

The modelling of resources is fundamental to the quantitative analysis. The nature of the necessary extension of the modelling language highly depends on the modelling style, which is applied to model the resources. The OMG proposal distinguishes between two ways of looking at the resource model. In the first, the so-called *peer interpretation*, a client and its used resource coexist at the same computational level. The *layered interpretation* is structurally very similar, but appears in a different context where a client (such as an application) is related to the resources that are used to implement it (such as the software and hardware environments used). Thus the client and the resource are not really co-existing, but rather two complementary perspectives of the same modeling constructs.

In realization layering (opposed to refinement layering) each level defines a distinct part of the system. The information in each layer is unique and the full system is only defined by the aggregate of all the layers. The lower layer defines a set of resources and resource services with offered QoS values (e.g., processor throughput, memory capacity, and communication bandwidth), which can be compared to the required QoS values of the elements in the upper layer. In the peer interpretation the offered and required QoS values can be compared using associations between the different components of the model, while in the layered one they can be compared along the dependencies. In both cases there are model elements representing activities (executions, transitions, steps), which have performance parameters, and other ones representing resources (active resources, interpreters, engineering model elements), which have dependability parameters.

As it was outlined in this section, the main question of automated model analysis is how to include supplementary annotations – required by the different analysis tools – in the UML model. Any approaches intended to be general to some extent have to be open for:

- different aspects of quantitative analysis,
- different analysis techniques with different underlying mathematical formalisms,
- different kinds of resource modelling.

4 Notation of QoS characteristics

The notation of QoS characteristics described in this work is based on the works of Svend Frølund and Jari Koistinen at the HP Labs, who proposed a general Quality-of-Service specification language (the QML) to capture QoS properties as part of the design of distributed object systems. Because of its generality and being object-oriented, QML can be applied in the extension of UML for quantitative analysis well. An adapted version of it could replace the "Tag Value Language" (TVL) of the OMG proposal, which was defined for specifying the value fields of tagged values for the representation of QoS characteristics.

In the first subsection we recall the main concepts of the QML terminology with respect to the quantitative modelling in UML as described in the OMG proposal. In the following subsections the application of these concepts in quantitative modelling is shown on some examples.

4.1 The QML terminology

In QML terminology, the lists of QoS constraints required or offered by a given object are called a *contract*. *Contract types* define the structure of its instances by containing a *dimension type* for each of its dimensions (e.g. throughput, reliability, failure masking, operation semantics etc.). In addition to simple constraints QML supports more complex (statistical) characterisations that are called *aspects* (like *mean*, *variance*, *frequency* and *percentile*).

The QML concept of profiles describes the QoS properties of services, which export a given set of operations and attributes. It is applicable in modelling for quantitative analysis only when there are special ser-

vices of objects included. In layered interpretation resources can offer resource services to their clients. The different ways in which QoS profiles can be bound to specific services (*binding*) will be discussed in the next section.
QML defines a conformance relation on profiles, contracts and constraints of the same type. This relation can be utilised to compare the offered and required quality of services. This is one of the advantages of adopting QML for quantitative modelling, because the OMG proposal does not specify the comparison of QoS values described in the TVL.

4.2 Contract types for performance modelling

The 8th chapter of the OMG proposal describes the approach and UML extensions required to perform basic performance analysis of UML models. Our work shows the QML-based approach on the QoS aspects, which were specified in this chapter of the proposal, but it can be applied on the other aspects as well. The description of the domain concepts of the performance analysis model in the OMG proposal (e.g. scenario, step, host, workload etc.) is only meant to be used as a basis for deriving similar UML stereotypes and tagged values. Therefore the QML contract types below can only serve as references as well. To minimise the possibility of confusion and conflict with other UML profiles, the OMG proposal prefixes all extension element names pertaining to this portion of the proposal with the "PA" prefix. In this work, this prefix is kept to name extension elements, which originate in the OMG proposal. In the proposal, the structures of the domain concepts are defined in an informal way, which can be formalised by defining a dimension type for each of them.
Our first examples will be contract types for the concepts of scenarios and steps:

```
type Scenario = contract{
   PAhostdemand: decreasing
      numeric ms;
   PArespTime: decreasing
      numeric ms;
}
```

The QML contract type "Scenario" describes the structure of the QoS characteristics of the concrete concept (used directly by the designer) of Scenario in the OMG proposal. As defined in the proposal, Scenarios have only two attributes: hostExecutionDemand and responseTime. Both characteristics (measures) have a numeric domain with the unit "ms" (milliseconds), and in both cases smaller values are better (both domains are decreasing). As it highly depends on the context, we leave the definition of the domains (especially the ordering and the unit) open, and state that the above definition is only for demonstration purposes. QML currently does not support different numeric domains, but it can be easily amended to distinguish between integer and real domains.
The OMG proposal defines both characteristics (hostExecutionDemand and responseTime) with a more complex domain type to allow the description of the type of value meaning, whether it is an average value, a variance, a k^{th} moment, a percentile range, a probability distribution or else. For this purpose QML has the concept of aspects, which will be introduced in the next subsection where QML contracts will be described.
The OMG proposal allows defining each performance measure in different versions in the same model. The possible versions of a measure may define a required, an assumed, a predicted (estimated) and a measured value. We propose not to include this information in the contract types, to support comparison of different versions (e.g. assumed and measured ones) of the same measures.
QML defines a conformance relation to provide rules for comparison of contracts of the same contract type. There are two possible ways to include the versions in the contract types:

- Definition of separate dimensions for each version (dimensions may be left unconstrained in the contracts), which would lead to a comparison of contracts without comparison of the different versions of the dimensions.
- Definition of a new enumeration-based dimension for indication of the version of the given contract. In this case either only contracts of the same version can be conform to each other, or an ordering of the different possible versions should be defined, which ordering cannot be general enough.

Our proposal is to include the versions in the binding, which means that different versions of the same measures are described in contracts of the same contract type and can be freely compared.
The contract type "Step" describes the more complex structure of the QoS characteristics of the Step concept in the OMG proposal. Besides having more numeric-domain dimensions than the original one (some of them without a unit) it deviates from the corresponding definition in the OMG proposal by skipping the characteristics "operations".

```
type Step = contract{
  PAdemand: decreasing numeric ms;
  PArespTime: decreasing
     numeric ms;
  PAprob: decreasing numeric;
```

```
    PAdelay: increasing numeric ms;
    PAextOp: increasing set {pos-
       sible resource operations};
    PAinterval: decreasing
       numeric ms;
}
```

The "operations" attribute of the step is defined by the OMG proposal to specify the set of operations of resources used in the execution of the step but which are not explicitly represented anywhere else in the model. The actual value of the "operations" attribute is defined by a string, which is used to identify an external operation and either

- the number of repetitions of that operation that are performed
- or a performance time value.

Contrary to the OMG proposal the QML strictly separates functional and quantitative properties of services. It defines a service specification to contain an interface and a QoS profile, the first one describing operations and attributes exported by the service, and the latter one describing QoS contracts for the attributes and attributes described in the interface.

We favour this separation of functional and quantitative properties, and are of the opinion that it conforms more to the multi-aspect nature of the UML, which provides separate diagrams for describing different views of the system under consideration. Therefore we prefer modelling operations in interface descriptions in class- and object-diagrams, assign QoS characteristics to them explicitly, and associating them to the given step.

For further examples two other QML contract types are shown below: "Host" and "OpenLoad".

```
type Host = contract{
  PAutilization: decreasing
     numeric;
  PAschdPolicy: enum {FIFO, HOL,
     PR, PS, PPS, LIFO};
  PArate: increasing numeric;
  PActxtSwT: decreasing numeric;
  PAprioMin: decreasing numeric;
  PAprioMax: increasing numeric;
  PApreemptable: increasing enum
    {TRUE, FALSE}
     with order {TRUE < FALSE};
  PAthroughput: increasing numeric;
}
```

The QML contract type "Host" applies two further concepts of QML, the enumeration and the ordering of domains (numeric domains are ordered per definitionem). The enumeration-based *schedulingPolicy* dimension (QoS characteristic) may be constrained to any single literal, which are listed above. The *isPreemptable* dimension can be constrained to *TRUE* or *FALSE*, and the above defined ordering implies that a non-preemptable resource may be conform to a preemptable client, since larger elements are stronger in increasing domains. In general, ordering is transitive. (To the question whether the *isPreemptable* dimension should be ordered, and if it should, then how, it is stated again that the definitions of orderings and units in this paper are only for demonstration purposes.)

In this contract type, the original *priorityRange* dimension is split into two simple numeric dimensions to describe the lower and upper end of the range. The QML specifies no range-based domains, however, it would be simple to define it based on the semantics of set-based ones.

```
    PAprioRange: increasing range
       numeric;
```

This way a range containing another range would be defined to be larger, and in an increasing range-based domain a larger range would be stronger.

```
type OpenLoad = contract{
  PArespTime: decreasing
     numeric ms;
  PApriotity: increasing numeric;
  PAoccurrence: increasing numeric;
}
```

The QML contract type "OpenLoad" applies no further concepts of QML, but it contains a dimension for the QoS characteristics *occurrencePattern*, which is associated with a more complex domain (RTarrivalPattern) in the OMG proposal. This dimension is defined to describe the pattern of inter-arrival times between consecutive instances of the start event depending on the nature of the series of intervals, but in QML's view it is practically nothing else but a numeric dimension, which may be constrained in unusual ways (see Section 4.3.1).

Here we avoid the description of the contract types for the other concepts of the OMG proposal, and rather discuss how to describe the QoS characteristics of a given system.

4.3 Contracts for performance modelling

In QML, to capture the structure of contracts within a given QoS category, a contract type specifies a dimension type for each dimension within the category. Contracts are instances of contract types. In general, a contract contains a list of constraints, and each con-

straint is associated with a dimension. A contract may specify constraints for all or only for a subset of the dimensions in its contract type. Omission of a specification for a particular dimension indicates that the contract is trivially satisfied along that dimension.
Constraints may be simple, containing of a name of a dimension, an operator and a value (a domain element). The QML uses the following set of operators: {==, <, <=, >, >=}. Inequality operators are allowed for ordered domains only. For decreasing domains {==, <=, <} are allowed, and for increasing ones {==, >=, >}. In the constraints, the domain of the allowed values is the domain specified in the corresponding dimension of the contract type.
In this paper, we do not discuss resources with multiple interfaces, and do not regard classes with explicitly listed set of exported methods. This way, even if we use the term "interface", a class, an object, or the unlisted set of methods exported by a class is meant. We radically simplify the semantics of QML to adapt it to the applied modelling techniques, however we keep the original terminology to keep the way open for extending our language for models with interfaces. In this context a QML profile does not consists of a subset of exported methods and a contract, but rather a class and a contract. However, the equivalence of the terms "class" and "interface" is a consequence of this simplification only.
A possible contract consisting of some simple constraints could be the following:

```
exampleHost = Host contract {
  PAutilization == 0.95;
  PAschdPolicy == FIFO;
  PAthroughput > 2;
}
```

The "exampleHost" contract an instance of the "Host" contract type, and it describes a particular quality of services. In this case, characteristics like the processing rate are unconstrained.
In an object-oriented setting, interfaces are typically subject to sub-typing or inheritance relationships. Since the interfaces (the services of the resources) can be defined through derivation and since there is a close coupling between QML profiles (contracts) and interfaces, the QML supports the derivation of profiles.

```
deriHost = exampleHost refined by {
  PAthroughput > 2.1;
  PArate > 1;
}
```

The "deriHost" contract is based on the "exampleHost" contract, but it is refined by some *stronger* constraints: The throughput must be larger, and processing rates equal to or smaller than 1 are no more satisfactory. The refinement of a contract may not contain less strong constraints than the original one, because of the inheritance of constraints. Contract refinement allows simple and consistent QoS-description of derived (specialised) resource and client objects.

4.3.1 Aspects

To specify constraints not only by a single value but in a more general way, the QML supports more complex statistical characterisations that are called aspects. The QML currently includes four generally applicable aspects: "mean", "variance", "frequency" and "percentile", but it allows user-defined ones as well. Based on the OMG proposal, some other aspects are defined.
In the above example the utilisation of a host is constrained to be exactly 0.95. In a more proper way it should be specified by a flexible description:

```
PAutilization {
   mean == 0.95;
   variance < 0.6;
}
```

It specifies that the measured values of the utilisation over some time period should have a mean of 0.95 and a variance less than 0.6. To avoid the unboundedness of such statistical characterisations it is useful to set limitations as well.

```
PAutilization {
   mean == 0.95;
   variance < 0.6;
   percentile 90 < 0.97;
   percentile 99 < 0.99;
   frequency (0, 0.5] > 30%;
}
```

This more precise constraint states additionally, that the strongest 90 and 99 percent of the measurements or occurrences should be less than 0.97 and 0.99 respectively, and that in more than 30% of the occurrences the utilisation should be larger than or equal to 0 and less than 0.5. (0 is included in the range but 0.5 is not. The use of open and closed boundaries is arbitrarily.)
Since the OMG proposal defines complex types (e.g. PAperfValue, RTarrivalPattern) to describe some of the QoS characteristics, other aspects are defined to implement these types.
The QML aspects *mean* and *percentile* are already known, *sigma*, *kth-mom* and *max* can be defined similarly. Aspects like the following ones can describe the standard probability distribution function values:

- **bernoulli ==** *real;*
- **binomialprob ==** *real;*
  binomialtrial == *integer;*
- **exponential ==** *real;*
- **gammak ==** *integer;* **gammamean ==** *real;*
- **geometric ==** *real;*
- **histogram1start ==** *real;*
  histogram1prob == *real;*
  histogram2start == *real;*
  histogram2prob == *real;*
  …
  histogram*n***start ==** *real;*
  histogram*n***prob ==** *real;*
  histogramendprob == *real;*
- **normalmean ==** *real;*
  normaldev == *real;*
- **poisson ==** *real;*
- **uniformstart ==** *real;*
  uniformend == *real;*

The set of actually applicable aspects can be highly dependent of the applied analysis tool. Aspects like the following ones can describe the arrival pattern values:

- **boundedmin ==** *real;*
  boundedmax == *real;*
- **burstyinterval ==** *real;*
- **burstymax ==** *integer;*
- **irregular1 ==** *real;*
  irregular2 == *real;*
  …
  irregular*n* **==** *real;*
- **periodic ==** *real;*
  periodicdeviation == *real;*

The aspects for probability distribution functions can describe unbounded arrival patterns.
Not all of the allowed statistical characterisations make sense for every dimension. The consistency of the aspects of a given constraint is the solely responsibility of the specifier, just like the accurateness of the chosen distribution functions and the correctness of the parameters.

4.3.2 Queries

If the UML description of the system under consideration is the front-end of the analysis, and the details of model analysis (including the specifics of its internal algorithms and data representation) are hidden from the modeller, there should be a way to specify queries. In the query the modeller describes the subject of the analysis, which will be passed to the analysis tool together with the quantitative data included in the model. In practice, of course, some the model. In practice, of course, some knowledge of the analysis tool and its techniques is both necessary and useful. Generally, there is no way to use an analysis tool without any knowledge about: what kind of results can be expected from it, what information is necessary for the analysis, and what types of specification of the input data can be used by the tool. There is no sense to feed data, which are described by exponential distribution functions, in a tool, which can only make analyses based on deterministic data.
The query require a set of values, which could not be specified by the modeller (at the time of the analysis, the modeller has no information to formulate constraints for the given dimension), therefore the query can be included in the contracts. This way, the domain of the result is specified unambiguously, and later it can be used for another analysis. Since neither the OMG proposal nor the QML specification discuss queries of this kind (the QML was specified with quite different goals), a new notation is specified. Any constrains may be specified with the new unary operator "**?**". The automatic transformation of the system model (Fig 1.) to a mathematical model has to formulate the syntactically correct question for the analysis tool (or report the erroneous query, if it is not interpretable for the given tool). The back-annotation has to replace the query with the result (or transform the error report).

```
queryHost = Host contract {
  PAutilization {
     mean == 0.95;
     variance < 0.6;
  }
  PAschdPolicy == FIFO;
  PAthroughput ?;
}
```

The "queryHost" contract describes a query about the throughput, while containing constraints for other dimensions. These constraints, together with other constraints specified on other components of the model can serve as the basis of the quantitative analysis. In this case, the domain of the result (and its possible unit) is defined by the "Host" contract type.

4.4 Conformance of contracts

Conformance allows comparing of two syntactically unrelated profiles. A profile *P* conforms to another profile *Q* if satisfaction of *P* also implies satisfaction of *Q*. (As in this paper interfaces are disregarded, "profiles" are equal to "contracts" here, because *P* conforms to *Q* if the contracts in *P* associated with an

interface entity *e* conform to the contracts associated with *e* in *Q*.)

Contract conformance is defined in terms of conformance for constraints, which defines when one constraint in a contract can be considered stronger or as strong as another constraint for the same dimension in another contract of the same contract type. Conformance for constraints for dimensions with ordered domains is based on ordering, for set-based dimensions it is based on the set inclusion relation. For unordered domains the conformance relation reduces to equality. If a profile contains a query, it is not conform to any other profiles.

For constraints including statistical characterisations (aspects like mean, percentile etc.) the conformance relation is based on aspect signatures. The aspect signatures of the predefined QML aspects and the formal definition of the conformance relations based on them are described in [10]. The signature of the aspects, which are defined in this work additionally, can be defined in the same way. For the sake of brevity, this details are not outlined here.

Conformance checking is to compare two different QoS, whether the quality offered by a given resource satisfies the quality required by a client. Being a simple yes/no question, this is not part of the traditional quantitative analysis. However, if there are many QoS descriptions in a system model, this task should be automated as well. A framework for modelling, which provides tools for the quantitative analysis, should be able to perform this simple rule-based comparison of structured data.

5 Binding

A contract defines the characteristics of a quality, in which system components may provide their services. Contracts (profiles) do not describe the QoS of a given component, they are formulated without any references to given components. The reference is first defined by bindings.

To define the QoS, that is required or offered by various components in the system, requires that a profile (contract) can be bound to a relation between a client and a resource object as part of the design.

In UML, the inter-object relations are described in class and object diagrams. An extension of the UML is defined in [8] to support the definition of QoS properties. This extended design notation fits the general requirements well, but it only covers the static structure diagrams. A similar notation can be used in deployment diagrams for layered models (i.e. in models, where the client and the resource are not really co-existing, but rather two complementary perspectives of the same modeling constructs).

These diagrams are suitable to describe the resulting QoS of the system components (the QoS requirements and offers), which cumulate from the QoS of the model elements. If there are interfaces in the model, then the cumulation is confined to the set of services, which is provided by the given interface. Analysing resulting QoS characteristics requires checking of profile conformance. Since the conformance relation is not symmetric, the asymmetric notation proposed in [8] and [10] is appropriate.

However, many of the quantitative characteristics describe properties of domain concepts, which are represented by interaction, activity or statechart diagrams. In this case, the individual QoS properties – which cumulate in the resulting values after the quantitative analysis of the model has provided the results – apply to model elements like collaborations, steps, states, activities and transitions.

For quantitative analysis the QoS properties are bound to the objects statically. The framework, which implements the automatic transformations to the applied analysis tool, has to provide the contract types for the modeller. The profiles (contracts) are defined textually using QML, and they either complement the UML model as an external document (since there are no appropriate constructs in the UML), or they are described by tagged values. The two kinds of representations may be combined in models. We prefer description of contracts (profiles) in such a way, that several model elements (with the same QoS characteristics) can reference the same contracts. Especially in the early phases of the design, when many design decisions are not yet made, the quantitative analysis (for comparison of possible architectures, algorithms and design concepts) is often based on typical values, which are the same for many model elements. In this case, common description of identical QoS properties facilitates:

- fast model construction,
- consistent modification of the values for testing different design concepts,
- consistent modification of the values for different kind of analyses of the same design.

If the profiles are described externally, binding can be represented:

- either by a tagged value, which associates the name of the chosen profile (contract) with the model element,
- or by a reference, which is drawn as a rectangle with dotted border within which the profile (contract) name is written. (This notation originates in the original QML specification.)

Having separate graphical entities for QoS profiles allows us to clearly show when the same profile is referenced to from multiple places. However, the

specification of the separate entity requires a significant addition to the UML metamodel.
For automatically generated QoS characteristics (e.g. back-annotation of analysis results), a separate description of contracts for each model element is more appropriate.

6 Conclusion

In this paper we have defined an object-oriented notation for modelling quantitative aspects. Our main goal was to support quantitative (performance, dependability, timeliness etc.) analysis of system models described in the Unified Modeling Language (UML). We supposed that the modelling of the system in consideration follows the UML Profile for Schedulability, Performance, and Time. In this work we have adopted the QoS Modeling Language (QML) for this context, which is different from its original one.
This way we have provided a notation, which can be tailored to a broad variety of quantitative modelling and analysis environments, depending on the main analysis goal, on the applied modelling techniques and tools.
We have shown that the QML is a proper language to extend UML model for quantitative analysis. We have outlined the adaptation end extension of the QML for describing the domain concepts, which are defined in the OMG proposal for performance modelling. Furthermore QML provides a uniform notation of QoS properties in the static and dynamic structure diagrams of the UML, and it extends traditional quantitative analysis by conformance checking. It can help in reducing visual clutter, which is one of the main common problems of the current proposals in this field.

Literature

[1] J. Rumbaugh, I. Jacobson and G. Booch: *The Unified Modeling Language Reference Manual* – Addison-Wesley Longman, Inc., Reading, Massachusetts, USA, ISBN 0-201-30998-X, 1999

[2] UML Semantics, Version 1.4 – Object Management Group, Needham, MA, USA, September, 2001, (www.omg.org/technology/documents/formal/uml.htm)

[3] A. Bondavalli, M. Dal Cin, D. Latella and Pataricza A.: *High-level Integrated Design Environment for Dependability (HIDE)* - In Proc. of Fifth Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS-99F), IEEE Computer Society Press, Los Alamitos, CA, USA, ISBN 0-7695-0616-X, pp. 87-92, November 18-20., Monterey, California, USA, 1999

[4] D. Harel: *Statecharts: A Visual Formalism for Complex Systems* – In Science of Computer Programming, 8(3), pp. 231-274, 1987

[5] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza and G. Savoia: *Dependability Analysis in the Early Phases of UML Based System Design* – International Journal of Computer Systems, Science & Engineering, **16**(5), pp. 265-275., September, 2001

[6] M. Dal Cin: *Structured Language for Specification of Quantitative Requirements* – In Proc. of the fifth IEEE Int. Symposium on High Assurance Systems Engineering (HASE 2000), IEEE Computer Society Press, Los Alamitos, CA, USA, ISBN 0-7695-0927-4, pp. 221-227, November 15-17., Albuquerque, NM, USA, 2000

[7] *UML Profile for Schedulability, Performance, and Time* – Object Manage-ment Group, Needham, MA, USA, June, 2001 (revised submission, OMG ad/2001-06-14, www.omg.org/cgi-bin/doc?ad/01-06-14)

[8] S. Frølund, J. Koistinen: *Quality of Service Specification in Distributed Object Systems Design* – In Proc. of the 4th USENIX Conf. on Object-Oriented Technology and Systems (COOTS), April 27-30., Santa Fe, New Mexico, USA, 1998

[9] S. Frølund, J. Koistinen: *Quality of Service Aware Distributed Object Systems* – In Proc. of the 5th USENIX Conf. on Object-Oriented Technology and Systems (COOTS), pp. 69-89., May 3-7., San Diego, California, USA, 1999

[10] S. Frølund, J. Koistinen: QML: *A Language for Quality of Service Specification* – HP Labs Technical Report (HPL-98-10), February, 1998