P²AMF: Predictive, Probabilistic Architecture Modeling Framework

Pontus Johnson[1], Johan Ullberg[1], Markus Buschle[1], Ulrik Franke[1,2], and Khurram Shahzad[1]

[1] Industrial Information and Control Systems,
KTH Royal Institute of Technology,
Osquldas v. 12, SE-10044 Stockholm, Sweden
{pj101,johanu,markusb,khurrams}@ics.kth.se
[2] FOI - Swedish Defence Research Agency,
SE-16490 Stockholm, Sweden
ulrik.franke@foi.se

Abstract. In the design phase of business and software system development, it is desirable to predict the properties of the system-to-be. Existing prediction systems do, however, not allow the modeler to express *uncertainty* with respect to the design of the considered system. In this paper, we propose a formalism, the *Predictive, Probabilistic Architecture Modeling Framework* (P²AMF), capable of advanced and probabilistically sound reasoning about architecture models given in the form of UML class and object diagrams. The proposed formalism is based on the Object Constraint Language (OCL). To OCL, P²AMF adds a probabilistic inference mechanism. The paper introduces P²AMF, describes its use for system property prediction and assessment, and proposes an algorithm for probabilistic inference.

Keywords: probabilistic inference, system properties, prediction, Object Constraint Language, UML, class diagram, object diagram.

1 Introduction

As an alternative to business and software service development by trial-and-error, it is desirable to predict the properties of envisioned services already in the early phases of the lifecycle. Such predictions may guide developers, allowing them to explore and compare design alternatives at a low cost. Business and software developers routinely argue for or against alternative design choices based on the expected impact of those choices on, e.g., the future system's efficiency, availability, security or functional capabilities. However, experience-based predictions made by individual developers have drawbacks in terms of transparency, consistency, cost and availability. Therefore, formal approaches to such predictions are highly desirable. In addition to prediction, system property analysis methods may be employed to assess properties of existing systems that are difficult to measure directly, e.g. in the case of information security. From an enterprise

M. van Sinderen et al. (Eds.): IWEI 2013, LNBIP 144, pp. 104–117, 2013.

interoperability perspective, one common approach to the field is the use of various forms of architecture models [1]. The abstraction in these models allows for quantitative reasoning about various issues. Incorporating the ability to perform quantitative analysis and prediction would further improve the reasoning. Most current system architecture frameworks, however, lack modeling languages that support interoperability analysis[2].

In this article, we present P^{2}AMF, a framework for generic business and software system analysis. P^{2}AMF is based on the Object Constraint Language (OCL), which is a formal language used to describe expressions on models in the Unified Modeling Language (UML) [3]. The most prominent difference between P^{2}AMF and OCL is the probabilistic nature of P^{2}AMF. P^{2}AMF allows the user to capture uncertainties in both attribute values and model structure.

1.1 OCL for System Property Predictions

In business and software development, many system qualities are worth predicting. These include theoretically well-established non-functional properties such as performance [4]. There are also properties where consensus on the theoretical base has yet to materialize, e.g. in the case of security [5], and interoperability [1]. Finally, there are many functional capabilities and non-functional properties that are so specific to a certain context that the analysis approach needs to be tailored for each instance, e.g. the coverage of the dictionary in a word processor application or the acoustic faithfulness of instruments in a music production application. The multitude of potentially interesting analyses prompts the need for generic languages and frameworks for system property analysis. An additional justification for such formalisms is the integrated analysis of multiple properties that they enable. Multi-attribute analysis provides a base for structured system quality trade-off, and the trade-off between different properties is a key element in any design activity.

To contain the analysis algorithms of multiple system properties, a framework needs to feature an appropriate and sufficiently flexible language. Many system property analysis approaches are based on logic, arithmetic operations and structural aspects of the system [6][7]. The dominating notation for software modeling today is the Unified Modeling Language (UML) [8]. Any generic framework for quality analysis therefore benefits from UML compatibility, allowing models to be shared between design and analysis.

The Object Constraint Language (OCL) [3], satisfies these requirements. OCL incorporates predicate logic, arithmetics and set theory, making it sufficiently expressive to contain most system property analysis needs. As a part of UML, OCL is also highly interoperable.

OCL was developed with *normative* purposes in mind, allowing the designer to constrain future implementation to conform not only to UML models, but also to OCL statements. However, OCL is also suitable for the *descriptive* (in particular *predictive*) purposes of system analysis, [5]. Still, one increasingly important characteristic of modern business and software systems is not captured by OCL: *uncertainty*.

As the business and IT-systems grows older, our knowledge of them becomes less certain. There are several reasons for this development. Firstly, business and software systems are rapidly increasing in complexity; they are growing in size as well as in the complexity of the underlying technologies. Secondly, as systems and components grow older, so do the people who developed them, and finally they will no longer be available. Combined with the poor state of documentation that plagues many projects, this adds to our uncertainty. Thirdly, the use of externally developed and maintained software is increasing.

To allow for explicit consideration of uncertainty in the analysis of non-functional properties, the framework presented in this paper, P^{2}AMF, is capable of expressing and comprehensively treating uncertainty in UML models. In P^{2}AMF, attributes are random variables. P^{2}AMF also allows the explicit modeling of structural uncertainty, i.e. uncertainty regarding the existence of objects and links. Indeed, as opposed to comparable formalisms (cf. Section 4 on related work), P^{2}AMF features probabilistic versions of logic, arithmetic and set operators, properly reflecting both structural uncertainty and the uncertainty of attribute values.

This article unfolds as follows: In Section 2, P^{2}AMF is described from the perspective of the user; in this section, the contribution of the article is provided in its most accessible form. The section also include references to some current applications, ranging from business aspects such as organizational structure to more IT related aspects. The most challenging part of the development of P^{2}AMF was the extension of OCL to a probabilistic context. The proposed inference approach is presented in Section 3. In Section 4, related work is considered. Finally, in Section 5 conclusions are described.

2 Introduction to P^{2}AMF

In this section, P^{2}AMF is described from the point of view of the user, i.e. an analyst evaluating a system property. In the first subsection, the differences between P^{2}AMF and the UML-OCL duo are explained. Then, an example class diagram is introduced and subsequently instantiated. This is followed by a subsection where the object diagram attribute values are predicted. The final two subsections describe the expressiveness and some current applications of P^{2}AMF.

2.1 Differences between P^{2}AMF and UML-OCL

The Object Constraint Language (OCL) is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled, or queries over objects described in a model [3].

From the user perspective, P^{2}AMF has many similarities to UML-OCL; from a syntax perspective, every valid P^{2}AMF statement is also a valid OCL statement. There are, however, also significant differences. The first and most important difference is that while OCL mainly is employed in the design phase to specify

constraints on a future implementation, P²AMF is used to reason about existing or potential systems. P²AMF may be employed to *predict* the uptime of a system while OCL is used to *pose requirements on the uptime* of the same system. While OCL is normative, mandating what *should* be, P²AMF is descriptive and predictive, calculating what *is* or *will be*.

A second difference between UML-OCL and P²AMF is the importance of the object diagram for P²AMF. As in standard UML, class diagrams with embedded expressions may be constructed that represent a whole class of systems. These diagrams may then be instantiated into object diagrams representing the actual systems considered. In P²AMF, however, the object diagrams become particularly significant as inference is performed on them.

Furthermore, P²AMF takes *uncertainty* into consideration. In particular, two kinds of uncertainty are introduced. Firstly, attributes may be stochastic. For instance, when classes are instantiated, the initial values of their attributes may be expressed as probability distributions. As will be described later, the values may subsequently be individualized for each instance.

Secondly, the existence of objects and links may be uncertain. It may, for instance, be the case that we no longer know whether a specific server is still in service or whether it has been retired. This is a case of object existence uncertainty. Such uncertainty is specified using an `existence` attribute that is mandatory for all classes. We may also be unsure of whether a server is still in the cluster servicing a specific application. This is an example of association uncertainty. Similarly, this is specified with an `existence` attribute on the association, implemented using association classes.

The introduction of two mandatory `existence` attributes and the specification of attribute values by means of probability distributions thus constitute the only changes to OCL as perceived by the user. These modest changes, however, allow for a comprehensive probabilistic treatment of the affected class and object diagrams, including both attribute uncertainty and structural uncertainty, enabling proper probabilistic inference over OCL expressions.

2.2 An Example Class Diagram

To illustrate the usage of P²AMF, consider the simple example of a cloud service. This is a case where the probabilistic nature of P²AMF is relevant; in cloud computing, the sheer complexity of the cloud mandate for an architecture, and architecture analysis, approach. Furthermore, there is a fundamental uncertainty about such things as the number of servers currently providing a given service, about the characteristics of these particular servers, etc. Nevertheless, these aspects influence the properties of the service at hand. From an interoperability perspective, properties such as response time and availability are important to consider. Although these are only a small part of aspects important for interoperability, they serve as an well-sized and self-contained example for illustrating P²AMF.

The class diagram for the example is given in Fig. 1. It contains three classes: `Service`, `Cloud` and `Server`. In the present example, we assume that the service

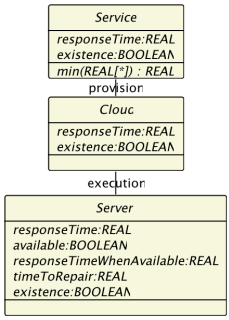

Fig. 1. An example class diagram

provider, in order to commit to a feasible service level agreement, would like to predict the future response time of the provided service. Thus, `responseTime` is an attribute of each of the three classes. Furthermore, every server can be up or down, thus prompting the attribute `available`. If a server is down, the time to repair is given by the attribute `timeToRepair`. Some of the attributes are given initial values while the rest are derived from other attributes. There is also a helper operation, `min`, returning the minimum of the provided values. Below, the model's P^2AMF expressions are provided.

```
context Service::responseTime:Real
derive: cloud.responseTime + min(cloud.server.responseTime)

context Service::min(values:Bag(Real)):Real
body: values->iterate(x:Real;
acc:Real=maxVal|x.min(acc))

context Service::existence:Boolean
init: Bernoulli(0.98)

context Provision::existence:Boolean
init: Bernoulli(0.98)

context Cloud::responseTime:Real
init: Normal(0.05, 0.01)

context Cloud::existence:Boolean
init: Bernoulli(1.0)

context Execution::existence:Boolean
init: Bernoulli(0.70)

context Server::responseTime:Real
derive: if available
then responseTimeWA
else timeToRepair
endif

context Server::responseTimeWA:Real
init: Normal(0.1, 0.02)

context Server::timeToRepair:Real
init: Normal(3600, 900)

context Server::available:Boolean
```

```
init: Bernoulli(0.95)

context Server::existence:Boolean
init: Bernoulli(0.97)
```

Going from the bottom and up in the P²AMF expressions above, first consider the Boolean server existence attribute. The probability that a given server exists is given by a Bernoulli distribution of 97%. Since the running example concerns a future state, this probability distribution represents the belief that a server will in fact be installed as planned, and will be dependent on the modeler's or other expert's knowledge . Continuing to the attribute `available`, the distribution specifies a 95% probability that a given server is up and running at any given moment. For the attributes `timeToRepair` and `responseTimeWA`, normal distributions specify the expected time (in seconds) before a server is up and running again after a failure and the response time for the case of a server that has not failed respectively. So far, we have considered four attributes assigned initial probability distributions on the class level. They thus represent the whole population of considered servers. Later, as the class diagram is instantiated, these estimates can be updated with system-specific data.

The top-most attribute of the `Server` class differs from the previously presented as it is *derived*. The derivation states that the response time of the server depends on whether it is available or not. If it is available, `responseTimeWA` gives the response time while `timeToRepair` returns the relevant value when the server is down. The `Execution` association connects the `Server` to the `Cloud` class. As there is uncertainty about whether a given server is connected to the Cloud, its `existence` attribute is assigned a probability of 70%.

The `Cloud` class has two attributes: its existence, which is similar to the existence attribute of the server class except that we are certain that the Cloud exists; and a real attribute with an initial probability distribution specifying the expected response time of the networking infrastructure. The `Provision` association class connects `Service` to the `Cloud`. Its features are similar to the `Execution` association class.

Finally, the class `Service` contains one derived attribute, `responseTime`, one operation, `min`, and the mandatory existence attribute. The service response time is given as a sum of the Cloud networking infrastructure response time on the one hand, and the minimum response time of the set of providing servers on the other. The `min` operation simply returns the minimum value of a set of values. The existence attribute is similar to those of the other classes.

2.3 An Example Object Diagram

The class diagram captures the general type of system and the causal effects that such systems are subject to. In order to make specific predictions, however, object diagrams detailing actual system instances are required. Instantiation follows the same rules as in object orientation in general. Classes are instantiated into objects, associations into links, multiplicities must be respected, and attributes may be assigned values (in the case of P²AMF, either deterministic values or probability distributions).

There is, however, one interesting and useful difference. In ordinary UML/OCL, values may not be assigned to derived attributes since those attributes are inferred from the derivation expression. Assignment of a different value than the one resulting from the derivation rule would lead to an inconsistent model. The probabilistic inference algorithm presented in Section 3, however, does allow the assignment of values to derived attributes, as long as attributes are assigned values within the ranges specified by the probability distributions, on the class level. The most useful consequence of this capability is the possibility to infer backwards in the causal chain. In our running example, we can therefore gain knowledge about the availability of the servers merely by observing the response time of the service. This capacity for backwards reasoning is not available in standard OCL/UML. As an example, consider a model where $x = y + z$. If x is assigned a value, OCL can tell us nothing of the value of y. P^{2}AMF, however, can. Therefore, in P^{2}AMF, all information that is provided in the object diagram is used to improve the predictions of attribute values.

Returning to the running example, consider the object diagram of Fig. 2. In this instance of the class diagram, the `calculator` – an instance of the `Service` class – uses `theCloud`, which is the single instance of the `Cloud` class. Three redundant `Server` instances are present in the Cloud, `calcServA`, `calcServB` and `calcServC`.

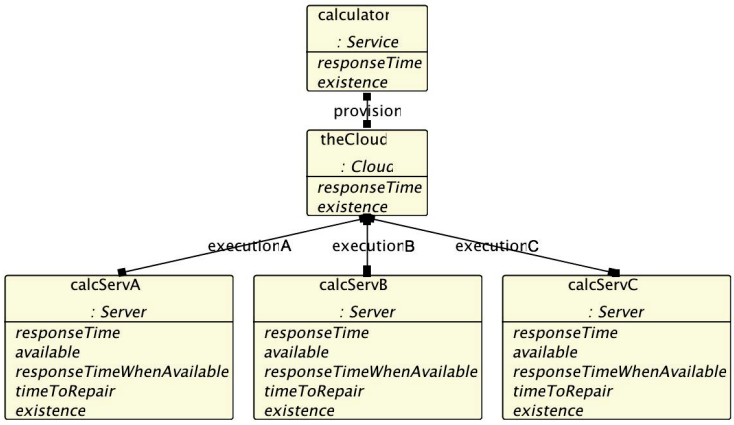

Fig. 2. Instantiation of the example class diagram

We assume that the service provider estimates the attribute values as presented in Table 1.Note that attributes may be assigned either deterministic values, as `theCloud.existence`, or stochastic ones, as e.g. `calcServC.timeToRepair`. Some are not assigned any values at all. These will instead be inferred as part of the prediction. Again, note that unlike standard UML/OCL, any attribute may be assigned a value and any attribute may be unassigned; inference will still be possible. A modeler can therefore obtain predictions based on the current state of knowledge, however poor that knowledge is. Of course, high uncertainties in the object diagram will generally lead to high uncertainties in the predictions.

Table 1. Attributes are assigned either probability distributions or deterministic values in the object diagram

Attribute type	Class.Attribute	Assigned value
Real	calculator.responseTime	
Boolean	calculator.existence	Bernoulli(0.997)
Boolean	provision.existence	True
Real	theCloud.responseTime	Normal(0.05, 0.005)
Boolean	theCloud.existence	True
Boolean	executionA.existence	Bernoulli(0.85)
Real	calcServA.responseTime	
Real	calcServA.responseTimeWA	Normal(0.08, 0.01)
Real	calcServA.timeToRepair	Normal(6000, 2000)
Boolean	calcServA.available	Bernoulli(0.94)
Boolean	calcServA.existence	Bernoulli(0.975)
Boolean	executionB.existence	Bernoulli(0.85)
Real	calcServB.responseTime	
Real	calcServB.responseTimeWA	Normal(0.03, 0.005)
Real	calcServB.timeToRepair	Normal(9000, 3000)
Boolean	calcServB.available	Bernoulli(0.91)
Boolean	calcServB.existence	Bernoulli(0.975)
Boolean	executionC.existence	Bernoulli(0.92)
Real	calcServC.responseTime	
Real	calcServC.responseTimeWA	Normal(0.12, 0.015)
Real	calcServC.timeToRepair	Normal(6000, 2000)
Boolean	calcServC.available	
Boolean	calcServC.existence	Bernoulli(0.975)

2.4 Inference in the Object Diagram

With support of a tool [9], the analyst can perform predictive inference on the object diagram described above with the click of a button. The details of the underlying algorithms are presented in Section 3. The results of the inference are new probability distributions assigned to the attributes. As these are typically non-parametric, they are most easily presented in the form of diagrams. Fig 3 displays the distribution of the most interesting attribute, `calculator.responseTime`. We note that the most probable response time is 80ms. This is the sum of the most probable response times of `theCloud` and `calcServB`, as `calcServB` is the fastest server and it is probably available. However, there is a certain probability (24%) that `calcServB` is down (i.e. that `available` is `false`) or that it is not in service (that `existence` is `false`). In this case, `calcServA` will most probably (83%) be available, and the response time will increase to 130 ms on average. If `calcServA` also fails or if it is not in service, `calcServC` will provide a mean response time of 170 ms. Despite the tripled redundancy, there is a small probability (1.2%) that none of the servers are available. In that case, the response time depends on the installed server with the shortest time to repair, i.e. either `calcServA` or `calcServC`, with a mean of 1:40h (6000 s) each. Finally, although quite unlikely, there is the risk (0,3%) that none of the servers will exist as modeled; they could have been taken out of service or were perhaps never installed in the first place. In this case, the response time will be so high that the exact value no longer matters.

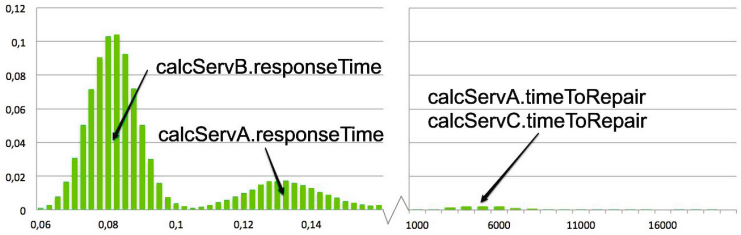

Fig. 3. `calculator.responseTime` probability distribution

As mentioned, backward inference is an important capability of probabilistic reasoning. As an example, suppose that when the system has been installed, an end user of the `calculator` service measures its response time to 130 ms. From this information, the prediction system automatically infers that both `calcServA` and `calcServB` must be either unavailable (90% probability) or non-existent (e.g. retired) (10% probability) while `calcServC` must be providing the service. This conclusion is reached automatically, but it can be understood intuitively as follows: Provided by redundant servers, the `calculator` service response time is given by the fastest available server. Since the measured service response time (taking the Cloud into account) is slower than those of `calcServA` and `calcServB`, they are surely down. Since the measured response time fits the probability distribution of `calcServC` when it is up and running, this must be the providing server.

2.5 Expressiveness of P²AMF

A set of expressive characteristics makes P²AMF particularly well suited for specifying predictive system property models. These include object orientation, support for first-order logic, arithmetics, set theory and support for expressing both class and instance level uncertainty, as described in this section.

P²AMF operates on class and object diagrams. The object-oriented features of such diagrams may therefore be leveraged by the predictive systems in P²AMF. These features are well known and include class instantiation, inheritance, polymorphism, etc. Secondly, P²AMF is able to express first-order logical relations. The predictive benefits of predicate logic are undisputed and used as a base for many deductive formalisms [10]. Furthermore, arithmetics, the oldest branch of mathematics, is used for prediction of properties ranging from hardware-related ones such as reliability [11] to organizational and economic ones, e.g. efficiency [12].

In order to efficiently make predictions on models such as the ones exemplified above, set theory is indispensable. The ability to speak of the number of components in a certain system, the qualities of a set of objects following a given navigation path in an object diagram, etc. are important for predictions on most systems with varying structure [10].

As previously discussed, for many real-world systems and situations, perfect information is rare. On the contrary, the available information is often incomplete or otherwise uncertain [13]. In P^{2}AMF, attributes of objects may be expressed by probability distributions. For many systems, not only the attribute values are associated with uncertainty, but also the system structure, e.g. does cloud service Z have double servers as the specification claims, or was one retired last month? The introduction of the `existence` attribute on classes and associations allows the specification of structural uncertainty in P^{2}AMF.

The object-oriented separation of theoretical prediction laws on the class level and the particulars about a specific system on the object level also pertains to the specification of uncertainty. The class-level modeler may need to express uncertainties about e.g. the strengths of attribute relations. For instance, to what extent a certain category of firewalls reduces the success rate of cyber attacks is rarely known precisely. Similarly as for the instance level, P^{2}AMF allows for specification of attribute uncertainty as well as structural uncertainty on the class level.

2.6 Applications of P^{2}AMF

P^{2}AMF has been used in class diagrams predicting such diverse properties as interoperability [1], availability [14], and the effects of changes to the organizational structure of an enterprise [15]. It has also been used for multi-property analysis [16]. These applications can be seen as evaluations of P^{2}AMF, in particular of the expressiveness of the formalism, as well as examples of the wide variety of properties that can be evaluated using P^{2}AMF. Furthermore, a software tool supporting modeling and prediction using P^{2}AMF has been developed, see [3] for a description of an early version of this tool.

3 Probabilistic Inference

In this section, we explain how inference is performed in P^{2}AMF models. A Monte Carlo approach is employed, where the probabilistic P^{2}AMF object diagram is sampled to create a set of deterministic UML/OCL object diagrams. For each of these sample diagrams, standard OCL inference is performed, thus generating sample values for all model attributes. For each attribute, the sample set collected from all sampled OCL models is used to characterize the posterior distribution.

Several Monte Carlo methods may be employed for probabilistic inference in P^{2}AMF models, including forward sampling, rejection sampling and Metropolis-Hastings sampling [17]. Of these, rejection and Metropolis-Hastings sampling allow the specification of evidence on any attribute in the object models while forward sampling only allows evidence on root attributes[1].

In this section, we will only present rejection sampling as it is the simplest method that allows evidence on all attributes. Let O^p denote a P^{2}AMF object

[1] Root attributes have no causal parents.

diagram, let $X_1, ..., X_m$ be the set of Boolean existence attributes $\mathbf{X}$ in such a diagram and let $Y_1, ..., Y_n$ be a topological ordering of the remaining attributes $\mathbf{Y}$ in the diagram. A topological ordering requires that causal parent attributes appear earlier in the sequence than their children[2]. The parents of Y_i, $\mathbf{Pa}_{Y_i}$, are those attributes that are independent variables in the OCL definition of the child attributes, $Y_i = f_{Y_i}(\mathbf{Pa}_{Y_i})$, where f_{Y_i} is the OCL expression defining Y_i. Furthermore, let $\mathbf{Y}^r$ represent the subset of $\mathbf{Y}$ that are root attributes, $\mathbf{Pa}_{Y_i^r} = \emptyset$, i.e. they are defined by probability distributions rather than by OCL expressions, $P(\mathbf{Y}^r)$. Let $\mathbf{Y}^{\overline{r}}$ represent the subset of $\mathbf{Y}$ that are *not* root attributes, $\mathbf{Y}^{\overline{r}} = \mathbf{Y} \setminus \mathbf{Y}^r$, i.e. that are defined by OCL statements rather than by probability distributions, $Y_i^{\overline{r}} = f_{Y_i^{\overline{r}}}(\mathbf{Pa}_{Y_i^{\overline{r}}})$.

The objective of the rejection sampling algorithm is to generate samples from the posterior probability distribution $P(\mathbf{X}, \mathbf{Y}|\mathbf{e})$, where $\mathbf{e} = \mathbf{e}^X \cup \mathbf{e}^Y$ denotes the evidence of existence attributes as well as the remaining attributes. The objective is thus to approximate the probability distributions of all attributes, given that we have observations on the actual values of some attributes, and prior probability distributions representing our beliefs about the values of all attributes prior to observing any evidence.

The first step in the algorithm is to generate random samples from the existence attributes' probability distribution $P(\mathbf{X})$: $\mathbf{x}[1], ..., \mathbf{x}[M]$. For each sample, $\mathbf{x}[i]$, and based on the P^{2}AMF object diagram O^p, a reduced object diagram, $N_i \in \mathbf{N}$, containing only those objects and links whose existence attributes, X_j, were assigned the value **true**, is created. Some object diagrams generated in this manner will not conform to the constraints of UML. In particular, object diagrams may appear where a link is connected to only one or even zero objects. Such samples are rejected. Other generated object diagrams will violate e.g. the multiplicity constraints of the class diagram. Such samples are also rejected. Finally, some OCL expressions are undefined for certain object diagrams, for instance a summation expression over an empty set of attributes. Remains a set of traditional UML/OCL object diagrams, $\mathbf{\Xi} \subset \mathbf{N}$, whose structures vary but are syntactically correct, and whose attributes are not yet assigned values.

In the second step, for each of the remaining object diagrams, Ξ_i, the probability distribution of the root attributes, $P(\mathbf{Y}^r)$ is sampled, thus producing the sample set $\mathbf{y}^r[1], ..., \mathbf{y}^r[size(\mathbf{\Xi})]$. If there is evidence on a root attribute, the sample is assigned the evidence value. Based on the samples of the root attributes, the OCL expressions are calculated in topological order for each remaining attribute in the object diagram, $y_i^{\overline{r}} = f_{y_i^{\overline{r}}}(\mathbf{Pa}_{y_i^{\overline{r}}})$. The result is a set of deterministic UML/OCL object diagrams, $\mathbf{\Lambda} \subset \mathbf{\Xi}$, where in each diagram, all attributes are assigned values.

The third step of the sampling algorithm rejects those object diagrams that contain attributes which do not conform to the evidence. The sampling process ensures that root attributes always do conform, but this is not the case for OCL-defined attributes. The final set of object diagrams, $\mathbf{O} \subset \mathbf{\Lambda}$, contains attribute

[2] That it is possible to order the attributes topologically requires the absence of cycles. Acyclicity is therefore a requirement for P^{2}AMF, just as for e.g. Bayesian networks.

samples from the posterior probability distribution $P(\mathbf{X}, \mathbf{Y}|\mathbf{e})$. These samples may thus be used to approximate the posterior. The algorithm is presented in pseudo code below.

```
for(int i=1; i<M; i++) {
x = sampleExistenceAttributes();
  x = sampleExistenceAttributes();
  N = extractObjectDiagram(O^p, x);
  if (syntacticallyCorrect(N)) {
   y = sampleRemainingAttributes();
   Λ = assignAttributesToDiagram(y, N);
   if (conformsToEvidence(Λ)) {
    O.add(Λ);
   }
  }
}
```

4 Related Work

There are three categories of work that in different ways are similar to P^{2}AMF. The first category includes variants of first-order probabilistic models. Among other proposals, these include Bayesian Logic (BLOG) [18] and Probabilistic Relational Models (PRM) [19]. These are similar to P^{2}AMF in their use of object-based templates which may be instantiated into structures amenable to probabilistic inference. However, they do not consider how logic and arithmetic operators are affected by structural uncertainty.

The second category of related work comprises query and constraint languages such as SQL [20] and OCL [3]. Similarly to P^{2}AMF, these languages allow logical and arithmetic queries of object or entity models. They are, however, deterministic rather than probabilistic.

The third and most important category of related work is work on stochastic quality prediction for software architecture. These include MARTE [4], KLAPER [21] and the Palladio component model for model-driven performance prediction [22], where two of them have opted for UML or MOF based modeling formalisms. However, common to all of these contributions is their focus on the analysis of particular properties. P^{2}AMF differs from these, as it does not propose specific analyses but rather provides a general language for expressing them. The closest match is probably the work by Ferrer et al. on multiple non-functional property evaluation [23], using the Dempster-Shafer approach to probabilistic reasoning. However, P^{2}AMF is more general still; aiming to offer not just a toolbox but a unified *language* where the best practice of e.g. reliability or performance modeling can be expressed. Within this third category, there are also generic frameworks for system quality analysis, such as ATAM [24]. These typically provide different support than P^{2}AMF, and are not based on probabilistic foundations.

5 Conclusions

Prediction and assessment of the expected quality and behavior of business and software systems already in the design stage is a desirable capability. With the frequent re-configurations of services in a complex and uncertain environment, the need for such analyses to deal with uncertainty grows.

In this paper, we have reported on a language and tool for probabilistic prediction and assessment of system properties. The formalism, P^{2}AMF, supports automatic probabilistic reasoning based on set theory, first-order logic and algebra. With class and object diagrams as a base, P^{2}AMF is compatible with UML. This paper has introduced P^{2}AMF and exemplified it for a simple analysis case, pointed out other areas where P^{2}AMF is being employed and described the algorithm for performing the required probabilistic inference.

References

[1] Ullberg, J., Johnson, P., Buschle, M.: A language for interoperability modeling and prediction. Computers in Industry (2012)
[2] Chen, D., Doumeingts, G., Vernadat, F.: Architectures for enterprise integration and interoperability: Past, present and future. Computers in Industry 59(7), 647–659 (2008)
[3] Object Management Group: Object Constraint Language. Version 2.3 (2010)
[4] Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version 1.0 (2009)
[5] Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
[6] Hansson, H., Jonsson, B.: A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
[7] Ritchey, R., Ammann, P.: Using model checking to analyze network vulnerabilities. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy, S & P 2000, pp. 156–165. IEEE (2000)
[8] Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4 (2011)
[9] Ullberg, J., Franke, U., Buschle, M., Johnson, P.: A tool for interoperability analysis of enterprise architecture models using Pi-OCL. In: Enterprise Interoperability IV, pp. 81–90 (2010)
[10] Spivey, J.M.: The Z notation: a reference manual. Prentice Hall International (UK) Ltd. (1992)
[11] Lyu, M.R.: Handbook of Software Reliability Engineering. Mcgraw-Hill (1996)
[12] Mason-Jones, R., Towill, D.R.: Total cycle time compression and the agile supply chain. International Journal of Production Economics 62(1-2) (1999)
[13] Aier, S., Buckl, S., Franke, U., Gleichauf, B., Johnson, P., Närman, P., Schweda, C.M., Ullberg, J.: A survival analysis of application life spans based on enterprise architecture models. In: Proc. 3rd International Workshop on Enterprise Modelling and Information Systems Architectures, EMISA 2009. Lecture Notes in Informatics, pp. 141–154 (2009)

[14] Franke, U., Johnson, P., König, J.: An architecture framework for enterprise IT service availability analysis. Software & Systems Modeling (2013)
[15] Gustafsson, P., Höök, D., Franke, U., Johnson, P.: Modeling the IT impact on organizational structure. In: Proc. 13th IEEE International EDOC Conference, EDOC 2009 (2009)
[16] Närman, P., Buschle, M., Ekstedt, M.: An enterprise architecture framework for multi-attribute information systems analysis. Software & Systems Modeling (2013)
[17] Walsh, B.: Markov Chain Monte Carlo and Gibbs Sampling (2004)
[18] Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., Kolobov, A.: Blog: probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI 2005, pp. 1352–1359. Morgan Kaufmann Publishers Inc. (2005)
[19] Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1300–1307. Morgan Kaufmann Publishers Inc., San Francisco (1999)
[20] Melton, J., Simon, A.: Understanding the new SQL: a complete guide. Morgan Kaufmann Publishers (1993)
[21] Grassi, V., Mirandola, R., Randazzo, E., Sabetta, A.: KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability. In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.) The Common Component Modeling Example. LNCS, vol. 5153, pp. 327–356. Springer, Heidelberg (2008)
[22] Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3–22 (2009)
[23] Ferrer, A.J.: Optimis: A holistic approach to cloud service provisioning. Future Generation Computer Systems 28(1), 66–77 (2012)
[24] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley Longman Publishing Co., Inc. (2003)