Inf Syst E-Bus Manage (2014) 12:595–622
DOI 10.1007/s10257-014-0241-8

ORIGINAL ARTICLE

An architecture modeling framework for probabilistic prediction

Pontus Johnson · Johan Ullberg · Markus Buschle · Ulrik Franke · Khurram Shahzad

Received: 7 June 2013 / Revised: 25 November 2013 / Accepted: 25 January 2014 /
Published online: 14 February 2014

Abstract In the design phase of business and IT system development, it is desirable to predict the properties of the system-to-be. A number of formalisms to assess qualities such as performance, reliability and security have therefore previously been proposed. However, existing prediction systems do not allow the modeler to express *uncertainty* with respect to the design of the considered system. Yet, in contemporary business, the high rate of change in the environment leads to uncertainties about present and future characteristics of the system, so significant that ignoring them becomes problematic. In this paper, we propose a formalism, the *Predictive, Probabilistic Architecture Modeling Framework* (P^2AMF), capable of advanced and probabilistically sound reasoning about business and IT architecture models, given in the form of Unified Modeling Language class and object diagrams. The proposed formalism is based on the Object Constraint Language (OCL). To OCL, P^2AMF adds a probabilistic inference mechanism. The paper introduces P^2AMF, describes its use for system property prediction and assessment and proposes an algorithm for probabilistic inference.

P. Johnson · J. Ullberg (✉) · M. Buschle · K. Shahzad
Department of Industrial Information and Control Systems,
KTH – Royal Institute of Technology, 100 44 Stockholm, Sweden
e-mail: johanu@ics.kth.se

P. Johnson
e-mail: pj101@ics.kth.se

M. Buschle
e-mail: markusb@ics.kth.se

K. Shahzad
e-mail: khurrams@ics.kth.se

U. Franke
FOI – Swedish Defence Research Agency, 164 90 Stockholm, Sweden
e-mail: ulrik.franke@foi.se

Keywords Probabilistic inference · System properties · Business properties · Prediction · Assessment · Object Constraint Language · UML

1 Introduction

Many business endeavors depend heavily on the development of information systems (Ross et al. 2006). For these to be successful, it is desirable not to develop by trial-and-error, but rather by predicting the properties of envisioned services as early as possible in the lifecycle (Kurpjuweit and Winter 2007). Such predictions may guide architects and developers, allowing them to explore and compare design alternatives at a low cost (van Sinderen et al. 2012). Designers routinely argue for or against alternative design alternatives based on the expected impact of those choices on, e.g., the future financial viability, availability, security or functional capabilities. However, experience-based predictions made by individual developers have drawbacks in terms of transparency, consistency, cost and availability (Cooke and Goossens 2004; Nielsen 1994). Therefore, it is desirable to have formal approaches to business and IT architecture property prediction. In addition to prediction, system property analysis methods may be employed to assess properties of existing systems that are difficult to measure directly, as in the case of for instance information security (Freiling 2008).

1.1 Contribution

In this article, we present P^{2}AMF, a framework[1] for generic architecture analysis of systems[2]. It has to be stressed that the term system here is not limited to IT systems alone. Rather, following the Oxford dictionaries, it refers to a "set of things working together as parts of a mechanism or an interconnecting network; a complex whole" (Oxford Dictionaries 2013). P^{2}AMF can be used to describe everything that can be expressed in terms of classes, their attributes and relations between these classes. P^{2}AMF is based on the OCL (Object Constraint Language 2010). 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.

One of the main purposes of performing such architecture analysis is to provide decision support. Seen from the perspective of a generic decision cycle, e.g. the OODA loop (Richards 2004), c.f. Fig. 1, the work presented in this article can aid in encoding observations in terms of scenarios and describing the as-is and various to-be architectures. In particular, P^{2}AMF can analyse these scenarios in the orientation phase of the OODA loop. This in order to facilitate rational decision-making based on analysis rather than subjective judgement.

[1] We use the term *framework* to denote a formal (system description) language and a set of inference rules (for attribute value assessment).

[2] An early version of P^{2}AMF was published as Probabilistic Imperative Object Constraint Language (Pi-OCL) in 2010 (Ullberg et al. 2010) While the general lines of thought remain the same, the early version was unnecessarily dependent on a formalism that was limited to exact probabilistic inference rather than more general approximate sampling algorithms, and lacked performance evaluation.

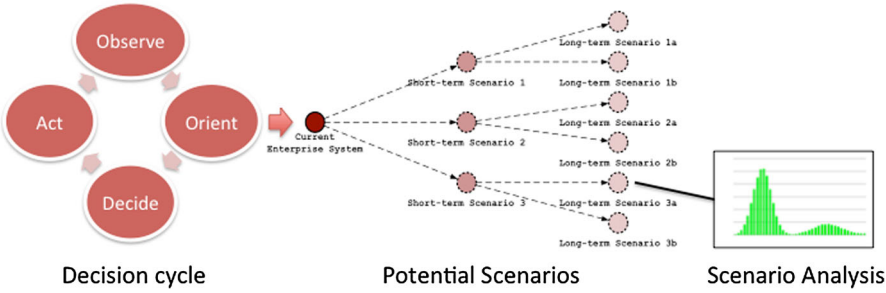

Fig. 1 The presented work seen from the perspective of the OODA loop

1.2 Architecture prediction

In Business and IT system design, many qualities are worth predicting. These include theoretically well-established non-functional properties such as performance, reliability and schedulability. For these properties, widely accepted analysis approaches have been established (UML Profile for MARTE 2009; Lyu 1996; Smith and Williams 2001). There are also properties where consensus on the theoretical base has yet to materialize, e.g. in the case of business network profitability (Gordijn et al. 2005; Johnson et al. 2013; Osterwalder 2004), security (Jürjens 2002; Lodderstedt et al. 2002; Sommestad et al. 2010), and interoperability (Chen and Doumeingts 2003; Ullberg et al. 2012). 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. compliance with local and national regulations for financial reporting. The multitude of potentially interesting analyses prompts the need for generic languages and frameworks for system property analysis. An additional *raison d'être* for such formalisms is the integrated analysis of multiple properties that they enable. Multi-attribute analysis provides a base for structured quality attribute trade-off, and trade-offs between different properties is a key element in any design activity.

To contain the analysis algorithms of multiple business and IT system properties, a framework needs to feature an appropriate and sufficiently flexible language. Many system property analysis approaches are based on logic (Halpern and Weissman 2003; Hansson and Jonsson 1994). There are also a number of quality analyses employing arithmetic operations (Joseph and Pandya 1986; Smith and Williams 2001). Finally, many business and IT property analyses require information on structural aspects of the system (role assignment, logical structure, deployment structure, etc.) (Gokhale 2007; Ritchey and Ammann 2000).

Although analysis is the most prominent feature of the framework, it should also act as the communicator of the design. It must therefore offer descriptive capabilities in terms of modeling. One of the most widely applied generic modeling languages today is the Unified Modeling Language (UML) (OMG Unified Modeling

Language 2011). Practically all major IT architecture and design tools are based on or support UML modeling (Lee et al. 2005; Quatrani 2002). Any generic framework for quality analysis therefore benefits from UML compatibility, allowing models to be shared between design and analysis.

The OCL (Object Constraint Language 2010), developed under the aegis of the Object Management Group (OMG), satisfies these requirements, including compatibility with UML. OCL incorporates predicate logic, arithmetics and set theory, making it sufficiently expressive to contain most system property analysis needs. OCL was developed with *normative* purposes in mind. However, OCL is also suitable for the *descriptive* (in particular *predictive*) purposes of architecture analysis, allowing the designer to predict the effects of various changes to a planned business or IT system, or to better understand the workings of an existing system. OCL has previously been employed for various types of analysis (Briand et al. 2003; Lodderstedt et al. 2002; Skene and Emmerich 2003). However, one increasingly important characteristic of modern business and IT systems is not captured by OCL: *uncertainty*.

1.3 The importance of uncertainty

As industries grow older, our knowledge of the business and IT systems being developed grows less certain. There are several reasons for this development. Firstly, the systems and the architectures describing them are rapidly growing more complex; they are growing in size as well as in the complexity of the underlying technologies. While a few decades ago, it was feasible for one person to fully grasp the workings of a company's IT architecture or its collaborations with suppliers and customers, this is no longer the case, simply due to the increased complexity. Secondly, as businesses and systems grow older, so do the people who designed them. The original developers of many systems and businesses running today have changed jobs, retired, or even died. Combined with the poor state of documentation that plagues many projects, this adds to our uncertainty. Thirdly, the use of externally developed and maintained business and IT services is increasing. Such uncertainty about e.g. the physical location of stored data may make an exit plan from a vendor or partner difficult to execute (Joint et al. 2009), thus resulting in a lock-in effect (Rold and Chamberlin 2011).

However, it is important to remember that the uncertainty inherent in the business yields not only risk, but also opportunity. Renowned consultancy Gartner argues that for organizations with highly variable total demand or uncertain future needs, cloud solutions, generally coupled with high uncertainties (Khajeh-Hosseini et al. 2010; Marston et al. 2011; Randles et al. 2010), outperform traditional methods, since the pay-per-use principle minimizes waste (Claunch 2011). However, to reap such potential business benefits, well-reasoned decisions under uncertainty must be made.

Arguably, these uncertainties are, in general, so significant that they ought not be ignored in the analysis. Whether a given prediction should be used for decision making or not depends on its credibility. In the face of uncertainty, the decision

maker must trade the expected consequences of inaccuracy against the cost of lowering it, typically through additional data collection and modeling. Due to its importance for decision making, it is standard practice in many areas, e.g. in the medical domain, to report on the credibility of analysis results and treatments (Altman et al. 1983; Daly and Bourke 2008; Gardner and Altman 1986). Such credibility is typically specified in terms of significance levels, confidence intervals or similar.

To allow for explicit consideration of uncertainty in the analysis of non-functional properties in business and IT, 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. Sect. 7 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.

1.4 Outline

This article is composed of eight sections. In Sect. 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 qualities that make P^{2}AMF suitable to different types of analyses are presented in Sect. 3; these qualities are put to the test in the business case analysis of Sect. 4. The most challenging part of the development of P^{2}AMF was the extension of the OCL inference mechanism to a probabilistic context. The proposed inference approach is presented in Sect. 5. Section 6 reports on the successful implementation of a tool for P^{2}AMF modeling and analysis. In Sect. 7, related work is considered. Finally, Sect. 8 contains the conclusions.

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 particular 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 OCL is a formal language used to state 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 (Object Constraint Language 2010). OCL expressions are written in the context of a class, an

association or an attribute in order to specify an invariant for one of these concepts. Starting from this context, an OCL expression can navigate through class diagram associations to produce collections of objects or attributes. Based on these, it is possible to evaluate various conditions, e.g. the existence of an object with specific properties, or comparing the value of an attribute with a threshold value.

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, P^{2}AMF is used to reason about existing or potential systems. P^{2}AMF 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 mainly normative, mandating what *should* be, P^{2}AMF is descriptive and predictive, calculating what *is* or *will be*.

A second difference between UML–OCL and P^{2}AMF is the importance of the object diagram for P^{2}AMF. As in standard UML, class diagrams with embedded expressions may be constructed that represent a whole class of concepts. These diagrams may then be instantiated into object diagrams representing the actual architectures using these concepts. In P^{2}AMF, however, the object diagrams become particularly significant as inference is performed on them.

Furthermore, P^{2}AMF 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 tuned 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 role in the organization is still in use 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 uncertain about whether a role is responsible for a process or if a server is still in the cluster servicing a specific application. These are examples 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 P^{2}AMF, consider the simple example of a cloud service. This is a case where the probabilistic nature of P^{2}AMF is relevant; in cloud

computing, the sheer complexity of the cloud mandates 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.

The class diagram for the example is given in Fig. 2. It contains three classes: `Service`, `Cloud` and `Server`. In the present example, we assume that the service provider 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. Such initial values may be either deterministic or probability distributions and correspond to the attribute value relevant for the whole class of concepts, and may be overwritten with more precise information for a specific instance in the object diagram. There is also a helper operation, `min`, returning the minimum of the provided values. Below, the model's P^{2}AMF expressions are provided. The class diagram also specify cardinality for the relationships. These are always specified deterministically as in a normal class diagram. However, since the existence of objects and links is uncertain, there might be object models that do not satisfy the cardinality constraints, which is managed by the sampling algorithm, see Sect. 5 for more information.

Going from the bottom up in the P^{2}AMF expressions below, 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`, truncated 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. Both are truncated at 0 to avoid the small probability for negative values. 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 %.

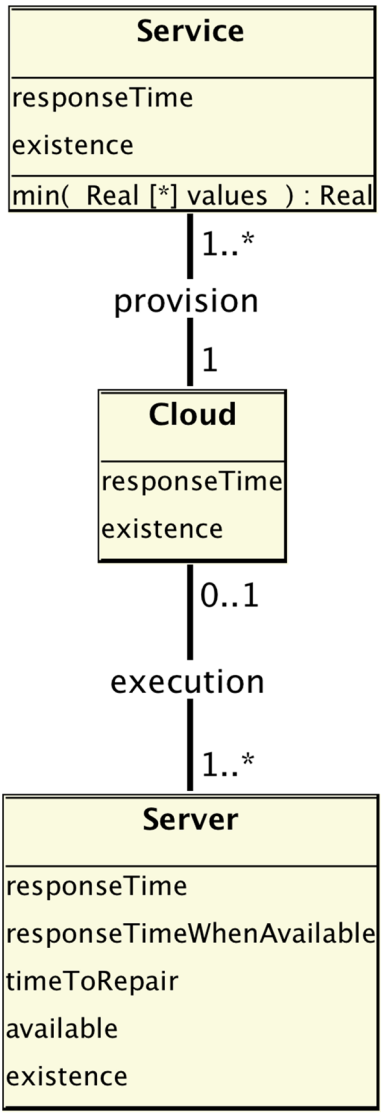

Fig. 2 An example class diagram

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 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.

```
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: TNormal(0.05, 0.01, 0)
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: TNormal(0.1, 0.02, 0)
context Server::timeToRepair:Real
  init: TNormal(3600, 900, 0)
context Server::available:Boolean
  init: Bernoulli(0.95)
context Server::existence:Boolean
  init: Bernoulli(0.97)
```

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. Instantiations 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 P^2AMF, 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 Sect. 5, 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^2AMF, however, can. Therefore, in P^2AMF, 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. 3. 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`.

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.

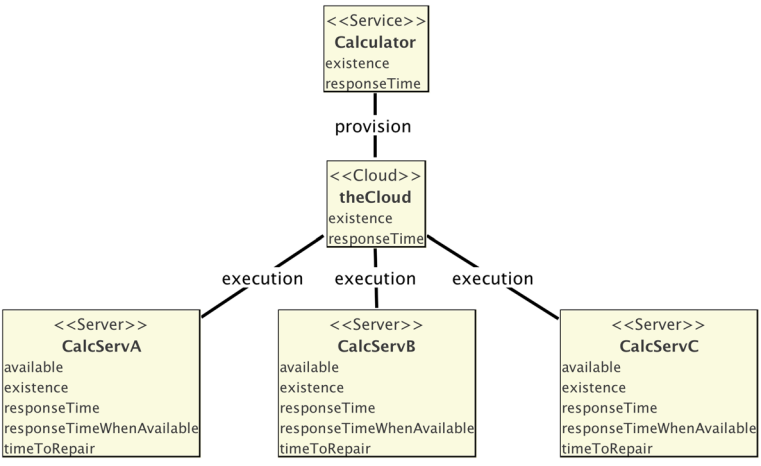

Fig. 3 Instantiation of the example class diagram

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	TNormal (0.05, 0.005, 0)
Boolean	theCloud.existence	True
Boolean	executionA.existence	Bernoulli (0.85)
Real	calcServA.responseTime	
Real	calcServA.responseTimeWA	TNormal (0.08, 0.01, 0)
Real	calcServA.timeToRepair	TNormal (6,000, 2,000, 0)
Boolean	calcServA.available	Bernoulli (0.94)
Boolean	calcServA.existence	Bernoulli (0.975)
Boolean	executionB.existence	Bernoulli (0.85)
Real	calcServB.responseTime	
Real	calcServB.responseTimeWA	TNormal (0.03, 0.005, 0)
Real	calcServB.timeToRepair	TNormal (9,000, 3,000, 0)
Boolean	calcServB.available	Bernoulli (0.91)
Boolean	calcServB.existence	Bernoulli (0.975)
Boolean	executionC.existence	Bernoulli (0.92)
Real	calcServC.responseTime	
Real	calcServC.responseTimeWA	TNormal (0.12, 0.015, 0)
Real	calcServC.timeToRepair	TNormal (6,000, 2000, 0)
Boolean	calcServC.available	
Boolean	calcServC.existence	Bernoulli (0.975)

2.4 Inference in the object diagram

With support of a tool (Ullberg et al. 2010), 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 Sect. 5. 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. Figure 4 displays the distribution of the most interesting attribute, `calculator.responseTime`. We note that the most probable response time is 80 ms. This is the sum of the most probable response times of `theCloud` and `calcServB`, as `calcServB` is the fastest server and 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`

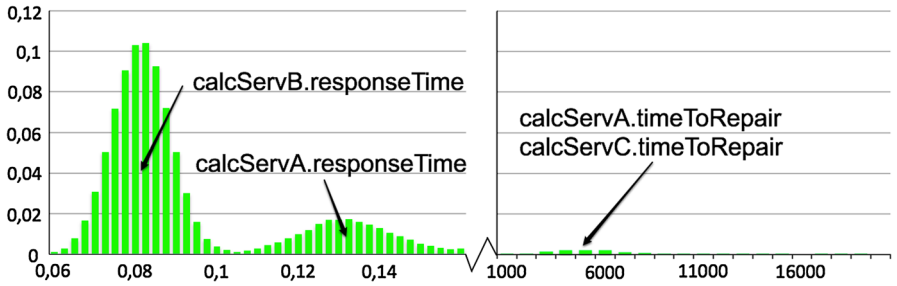

Fig. 4 `calculator.responseTime` probability distribution

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 (6,000 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.

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.

3 Expressiveness of P^{2}AMF

A set of expressive characteristics featured by P^{2}AMF make it 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. This section expands on these capabilities and the next section illustrate them in context.

3.1 Object orientation

P^{2}AMF operates on class and object diagrams. The object-oriented features of such diagrams may therefore be leveraged by the predictive systems in P^{2}AMF. These

features are well known and include class instantiation, inheritance, polymorphism, etc. The benefits include the following:

Real-world modeling The object-oriented paradigm has consistently proven suitable for modeling the real world.
Software system modeling Equally successful as in the modeling of the real world has the object-oriented paradigm been in modeling the software systems that operate on that world.
Instantiation From the point of view of prediction, the concept of instantiation allows a clear differentiation between the general prediction theory (expressed on the class level) and the specific instances of prediction (represented by object models). This separation is also a separation of concerns between the developer of the predictive system (e.g. an expert on some quality attribute) and its user (the modeler of a specific system).

Due to object-orientation's suitability for system analysis, several object-oriented prediction systems have previously been proposed (UML Profile for MARTE 2009; Grassi et al. 2007; Smith and Williams 2001).

3.2 First-order logic

P^2AMF is able to express first-order logical relations. The predictive benefits of predicate logic are undisputed. These are used as a base for many deductive formalisms (Jackson 2002; Spivey 1992), including those aimed at for instance information security (Halpern and Weissman 2003), interoperability (Allen 1997), and correctness (Moriconi and Qian 1994).

3.3 Arithmetics

If predicate logic is important for predictive systems, arithmetics, the oldest branch of mathematics, is perhaps even more so. Arithmetics is used for prediction of properties ranging from hardware-related ones such as throughput (Smith and Williams 2001), reliability (Lyu 1996) and execution time (Puschner and Koza 1989) to organizational and economic ones such as cost (Drury 2007), efficiency (Mason-Jones and Towill 1999) and value (Rappaport 2000).

3.4 Set theory

The object-oriented concept of instantiation allows the creation of many objects of the same class. In order to efficiently make predictions on such models, 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. The many prediction-oriented software specification formalisms based on set theory (Jackson 2002; Spivey 1992) testify to its relevance.

3.5 Instance-level uncertainty

As previously discussed, for many real-world systems and situations, perfect information is rare. On the contrary, the available information is often incomplete, old, vague, conflicting or otherwise uncertain (Aier et al. 2009). In P^{2}AMF, attributes of objects, e.g. the uptime of a certain server, the age of a piece of hardware, or the efficiency of an employee, may be expressed by probability distributions.

For many systems, not only the attribute values are associated with uncertainty, but also the system structure. Is it still the case that system X communicates with system Y? Does cloud service Z have double servers as the specification claims, or was one retired last month? Structural uncertainty grows important as the system grows larger and moves further from the modeler, e.g. into the Cloud. The introduction of the `existence` attribute on classes and associations allows the specification of structural uncertainty in P^{2}AMF.

3.6 Class-level uncertainty

As mentioned previously, the object-oriented separation of class and object diagrams by instantiation is particularly suitable for predictive models as the generic, theoretical prediction laws and structures may be captured on the class level while particulars about a specific system are left to the object model. This division also pertains to the specification of uncertainty. While the object-level modeler may be uncertain about the structure and attributes of a specific system model, 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. Uncertainty regarding such information may be codified by means of class-level attribute uncertainty in P^{2}AMF. Similarly as for the instance level, the `existence` attribute also allows specification of structural uncertainty on the class level.

4 A business case analyzed with P^{2}AMF

In order to show the usefulness of P^{2}AMF for business-related analysis, this section offers an applied example. Since IT is increasingly being procured "as a service", the example will address profitability analysis of service level agreements, SLAs, from the perspective of the service provider. In particular, we focus on availability provision, an important but difficult area of SLA writing.

Specifically, assume that we are about to sell an IT service to a customer. Following negotiations, there is now an SLA proposed. It sets an `Availability-Requirement`, expressed as a percentage, and also a requirement on the Time To Recovery, `TTRRequirement`, expressing the customer demand that recovery in the case of outage does not exceed h hours. To enforce these requirements, the SLA contains provisions on fines in case of breaches: There is an `AvailabilityFinePerBasisPoint` which has to be paid if the average annual availability goes below the requirement. For example, with a requirement of 99.50 % availability, a result of

99.35 % would entail 15 times the fine per basis point to be paid. Similarly, there is a a `TTRFinePerHour` which has to be paid if the TTR exceeds the requirement. For example, with a requirement of 4 hours recovery time, a recovery lasting 6 hours would entail twice the fine per hour to be paid. Finally, of course, the SLA contains a `SalesIncome`, i.e. the amount for which we sell it. But will it be profitable?

To find the `NetIncome`, i.e. the `SalesIncome` minus our costs for the IT service offered and any fines, we need to model our `ApplicationServiceOffer` in greater detail. As service providers, we have access to certain information that the customer does not. In particular, we know the `Cost` of our `ApplicationService`, as well as the distribution of its Time To Failure, `TTF`. For the sake of the example, we let this distribution be Weibull, following (Schroeder and Gibson 2010) and (Heath et al. 2002). Similarly, we know the the `Cost` of our `RecoveryProcess`, as well as the distribution of its Time To Recovery, `TTR`. For the sake of the example, we let this distribution be log-normal, following (Schroeder and Gibson 2010) and (Franke et al. 2013b).

Before proceeding with the example, it is worth to dwell on how this problem would be addressed manually, without P^{2}AMF. Probably, the business analyst would apply the well-known equation availability (1):

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \tag{1}$$

Using *mean* TTF and TTR, this equation gives a valid result for the steady state availability. This is a single figure—the mean—not the entire distribution. But—as pointed out by Snow and Weckman –it is dangerous to base conclusions about SLA breaches on means (Snow and Weckman 2007). P^{2}AMF allows us to consider the entire distribution of both availability and TTR. The result of a simulation (with suitable parameters) is given in Fig. 5. As is evident, the SLA at hand has a good chance of being profitable. However, there is also a substantial risk that it will run at

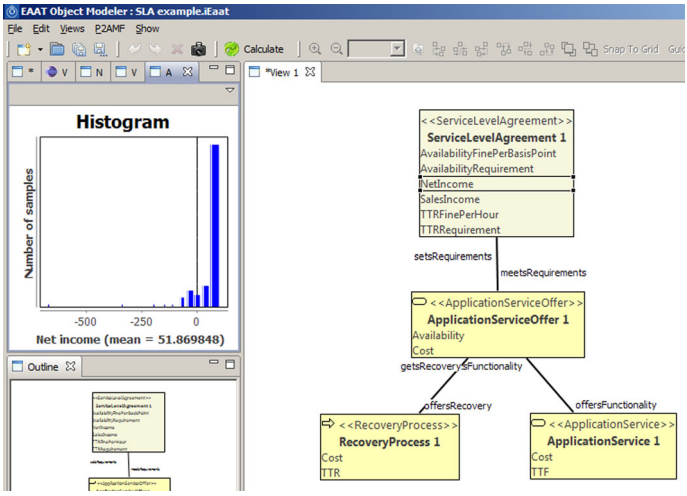

Fig. 5 `ServiceLevelAgreement.NetIncome` probability distribution

a loss, and a small chance that it will run at a considerable loss. The mean value (also displayed in the figure) gives no insight into these finer details of the opportunities and risks involved.

Briefly considering the P^{2}AMF implementation, the main work is done in the definition of the `NetIncome`-attribute:

```
let fine1 : Real =
  if self.meetsRequirements.Availability <
    self.AvailabilityRequirement
  then
    self.AvailabilityFinePerBasisPoint*
      (self.AvailabilityRequirement-self.meetsRequirements.Availability)*10000
  else
    0
  endif
in

let fine2 : Real =
  if self.meetsRequirements.offersRecovery.TTR >
    self.TTRRequirement
  then
    self.TTRFinePerHour*(self.meetsRequirements.offersRecovery.TTR
      -self.TTRRequirement)
  else
    0
  endif
in
self.SalesIncome-self.meetsRequirements.Cost-fine1-fine2
```

The two if-clauses (illustrating first-order logic) check to see if any of the fines for excessive downtime or availability are to be calculated. If so, the amounts are calculated (illustrating arithmetics) based on the values of the `Availability` and `TTR` attributes of the `ApplicationServiceOffer` and the `RecoveryProcess`, respectively. These are stochastic, based on sampling of Weibull TTF and log-normal TTR (illustrating instance-level uncertainty). In the final calculation, the stochastic fines and the fixed costs of the provider are both subtracted from the fixed income gained from selling the service, thus yielding the stochastic net income illustrated in the histogram in Fig. 5.

Indeed, this model can help us not only with the immediate decision of whether to accept or decline a concrete SLA on offering. It can also help us make more strategic decisions on availability. There are two ways of increasing availability: increase TTF or decrease TTR (Franke 2012). The P^{2}AMF model enables us to see how much it is worth to pay for better hardware or more robust software (which will increase TTF), and how much it is worth to pay for a stand-by repair crew (which will decrease TTR), given the SLAs that we have currently signed with our customers.

Improving business analysis of availability SLAs is important. In Franke et al. (2013c) practitioners explain that their companies are immature when it comes to SLA writing, particularly when IT services are delivered in complex architectures

with many layers of sub-contractors. Furthermore, recent research suggests that practitioners often fail to maximize expected utility when faced with availability SLA decisions (Franke et al. 2013a).

4.1 Other applications and examples

P^{2}AMF has been used to predict such diverse properties as availability (Franke et al. 2013c), interoperability (Ullberg et al. 2012), cyber security (Holm et al. 2013) and the effects of changes to the organizational structure of an enterprise (Gustafsson et al. 2009). It has also been used for multi-property analysis (Närman et al. 2012) of aggregated systems and services and tradeoffs between different attributes including cost and availability (Österlind et al. 2013). The papers cited contain other applied examples of P^{2}AMF that can be read by the interested reader. In total, more than 40 analyses have so far been conducted using P^{2}AMF. For example, P^{2}AMF has been used for analysis by more than 20 practitioners, either on their own or with support from the authors. Most of these cases belong to the financial, defense or power utility businesses. In most cases, the practitioners used the tool to do availability analysis, followed by cost evaluations and data accuracy analysis. Furthermore, P^{2}AMF as encapsulated in the tool has also been used to teach students enterprise architecture analysis. Throughout 5 courses, more than 50 students have used the tool to date.

5 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 (Koller and Friedman 2009; Walsh 2004). 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[3]. In this section, we will present rejection and Metropolis-Hastings sampling since evidence on all attributes is a likely scenario.

Both sampling algorithms have in common that the first step is to generate random samples from the existence attributes' probability distribution $P(\mathbf{X}) : \mathbf{x}[1], \ldots, \mathbf{x}[M]$. 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 extracted. Some object models generated in this manner will not conform to the constraints of

[3] Root attributes are attributes that have no causal parents.

UML. In particular, object models may appear where a link is connected to only one or even zero objects. Such samples are rejected. Other generated object models will violate e.g. the multiplicity constraints of the class model. Such samples are also rejected. Additionally, some OCL derivations are undefined for certain object models, for instance a summation derivation over an empty set of attributes. After this rejection procedure, a set of traditional UML/OCL object diagrams remains, $\boldsymbol{\Xi} \subset \mathbf{N}$. The structures in $\boldsymbol{\Xi}$ vary but all elements are syntactically correct. The attributes are not yet assigned values.

5.1 Rejection sampling

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 observations on the actual values of some attributes, and prior probability distributions representing beliefs about the values of all attributes prior to observing any evidence.

Algorithm 1: The rejection sampling algorithm.

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

The algorithm is depicted in Algorithm 1. Rejection sampling requires the attributes that are part of a sample $Y_1, \ldots, Y_n$ to be sorted in topological order i.e. parent attributes appear earlier in the sequence than the attributes that are calculated based on them, their children. Following the general first step, 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 $\mathbf{y}^r[1], \ldots, \mathbf{y}^r[size(\boldsymbol{\Xi})]$. 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, $y_i^{\bar{r}} = f_{y_i^{\bar{r}}}(\mathbf{Pa}_{y_i^{\bar{r}}})$. The result is a set of deterministic UML/OCL object diagrams, $\boldsymbol{\Lambda} \subset \boldsymbol{\Xi}$, where in each diagram, all attributes are assigned values.

The third step of the rejection 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 samples from the posterior probability distribution $P(\mathbf{X}, \mathbf{Y}|\mathbf{e})$. These samples may thus be used to approximate the posterior.

5.2 Metropolis-Hastings sampling

Metropolis-Hastings sampling (Koller and Friedman 2009; Walsh 2004) is an iterative sampling technique converging to a desired distribution limit. It aims to create a Markov chain MC with a stationary distribution being the desired distribution, i.e., a chain of samples where the sampled attribute values match the specified evidence. The algorithm is described in Algorithm 2. First one valid sample is created using rejection sampling. Once this sample is found it is used as the first element in the Markov chain.

Algorithm 2: The Metropolis-Hastings sampling algorithm.

```
Λ_init = rejectionSampling(M = 1)
MC.add(Λ_init)
for(int i=1; i<M+B; i++) {
  Λ = MC.getLast();
  Λ' = generateNewSample(Λ);
  P(Λ'|Λ) = calculateProbability(Λ', Λ);
  α = calculateProbabilityOfAcceptance(Λ', Λ, P(Λ'|Λ));
  if (α < l) {
    MC.add(Λ')
  }
  else {
    MC.add(Λ)
  }
}
removeBurn-InSamples(B)
```

The second step is to create a new chain element based on the last added element. A new sample is created as a copy of the last chain element. For the attributes without any specified evidence new values are generated using a candidate-generating distribution. Then the likelihood of the new sample given the old sample $P(\mathbf{x}'|\mathbf{x})$ is evaluated. Thereafter the probability of acceptance $\boldsymbol{\alpha}$ of the sample is calculated, considering the likelihood $P(\mathbf{x}'|\mathbf{x})$, which over time is given more weight to. If $\boldsymbol{\alpha}$ is greater than a given limit l the sample is added to the chain otherwise the last added element is added again. The second step is repeated until a predefined number M of chain elements has been added. The first samples are typically not used to evaluate the model; they are called burn-in samples B and train the algorithm. As a final step the burn-in samples are removed.

Similar to rejection sampling, Metropolis-Hastings sampling allows specifying evidence for any attribute of the model. This algorithm does need a comparably

smaller number of samples and is therefore more effective, especially when considering models including a large number of attributes. The biggest disadvantage of Metropolis-Hastings sampling is that, especially for models with many local minima, a solution not being the best one might be found. This is because of the chain structure of the result, where samples are based on their predecessor.

6 Implementation of P^{2}AMF

In this section, we report on a software tool that allows modeling of both probabilistic class diagrams and probabilistic object diagrams[4]. It also performs inference as described in Sect. 5. A complementary presentation of the tool is available in Buschle et al. (2013).

6.1 Design

The presented tool is implemented in Java using the Eclipse rich client platform. To provide the modeling facility the Eclipse Modeling Framework (EMF) (Steinberg et al. 2008) is used and extended. The tool is divided into two components, the Class Modeler, and the Object Modeler, corresponding to two file types: class and object diagrams.

The Class Modeler is a graphical editing tool for probabilistic class diagrams. In addition to the basic editing functionality provided by other class diagram modeling tools, the Class Modeler (1) allows attribute values defined either by probability distributions or by OCL expressions, (2) requires a value for the mandatory existence attributes of classes and associations, and (3) provides OCL syntax checking support using the OCL plugin of the Eclipse Modeling Framework Eclipse Modeling Framework (2011). A screen shot of the Class Modeler is presented in Fig. 6.

The Object Modeler (cf. Fig. 7) has two components: (1) an editing tool for probabilistic object models, and (2) an inference engine. The editing tool differs from other object diagram editor tools mainly in (1) allowing probabilistic attribute values, including the mandatory existence attributes, (2) displaying histograms for all attributes representing their probability distributions after inference, and (3) offering different inference algorithms and parameters. By the click of a button, the calculations described in Sect. 5 generate posterior probability distributions for all attributes.

6.2 Usage and performance

As mentioned, the tool has been used for modeling and prediction of several system properties. The largest class diagrams created in these projects have reached sizes of some twenty classes and sixty attributes. As object diagrams can grow significantly larger, we have produced examples with some seventy objects and five hundred attributes.

[4] The tool is available for download at http://www.ics.kth.se/eaat.

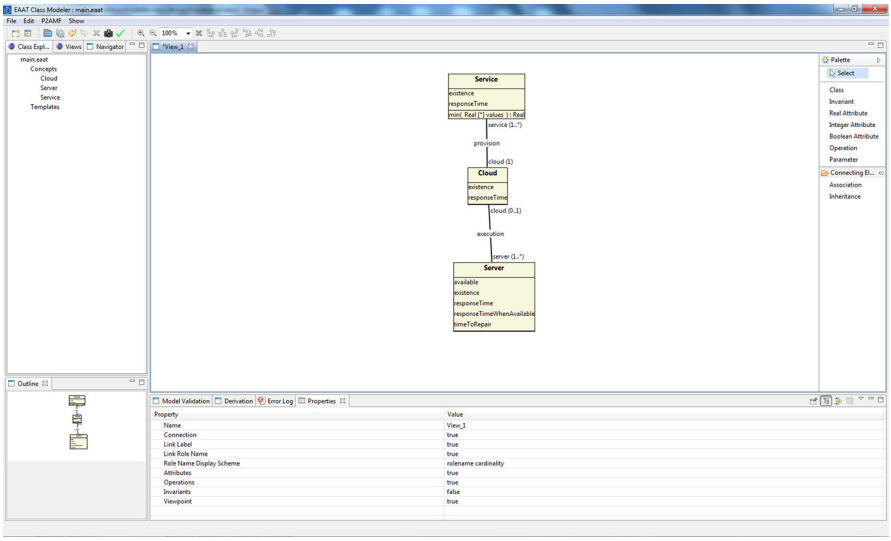

Fig. 6 Screen shot of the CLASS MODELER

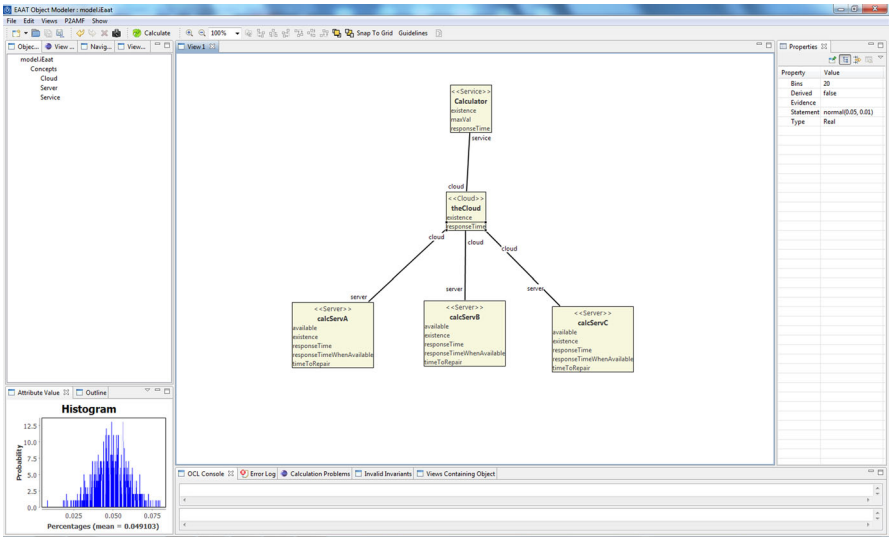

Fig. 7 Screen shot of the OBJECT MODELER

While model editing of these sizes is straightforward, the performance in the inference can be an issue, depending on the complexity of the P^{2}AMF expressions, the selected inference algorithm and its parameters. For our most complicated models, one Monte Carlo sample requires 0.5 s for the OCL inference on a standard

laptop.[5] For all algorithms, the inference time grows linearly with the number of samples. Figure 8 displays the inference time as a function of the number of samples for the example presented in Fig. 3. Note that 1,000 samples normally is a sufficient sample size; in the diagram, we display the results for larger sample sizes to highlight the linear relationship. For rejection sampling, the acceptance rate (the share of samples that conform to evidence and are thus not rejected) influences inference time inversely proportionally. Figure 9 shows the inference time per sample as a function of the inverted acceptance rate, also for the example of Fig. 3. Thus, the more unlikely the evidence, the greater the share of samples that is

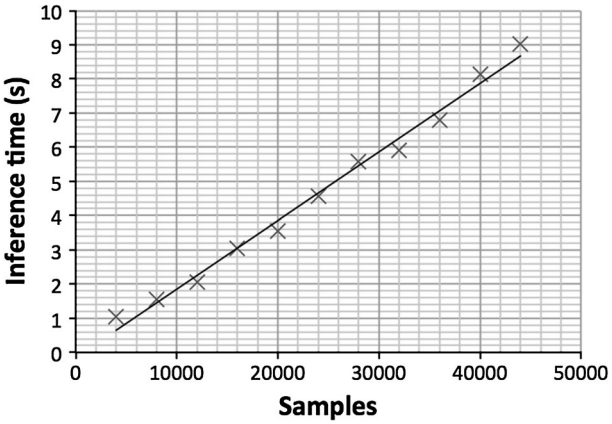

Fig. 8 Inference time as a function of the number of samples for the example of Fig. 3. 1,000 samples is normally a sufficient sample size

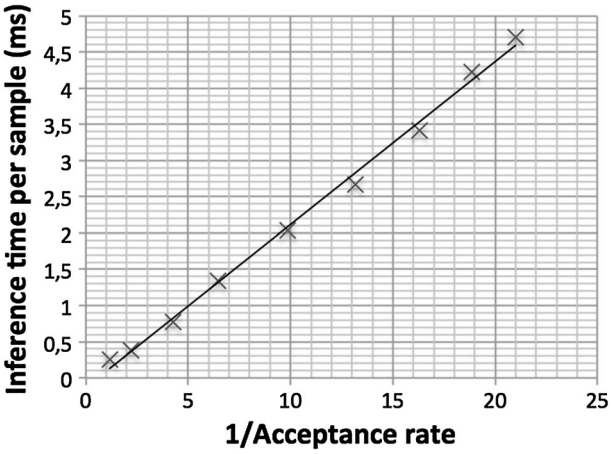

Fig. 9 Inference time per sample as a function of the inverted acceptance rate of Fig. 3

[5] MacBook Pro, 2.4 GHz Intel Core i5, 4 GB RAM.

rejected, and the longer the computation required to produce a specific number of samples. For the Metropolis-Hastings algorithm, inference time is also affected by choice of proposal distribution and burn-in time. Overall, the current implementation is usable but can encounter performance problems for certain models. Forward sampling for 1,000 samples in our largest models on the aforementioned hardware requires some 80 seconds to calculate. Compared to the time required for modeling (counted in hours or days), this is quite acceptable.

There are several options available for performance improvements, including more efficient coding, multithreading (which is expected to improve performance significantly) and even high-performance cloud computing.

7 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) (Milch et al. 2005), Probabilistic Relational Models (PRM) (Friedman et al. 1999) and Directed Acyclic Probabilistic Entity-Relationship (DAPER) models (Heckerman et al. 2004). 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, first-order probabilistic models also differ from P^{2}AMF; most importantly, they do not consider how logic and arithmetic operators are affected by structural uncertainty. Consider the following expression,

```
context
Person::avgAgeOfFriends:Real
  derive:
friend.age->sum()/friend->size()
```

In contrast to the mentioned works, P^{2}AMF will properly weigh the friends' probability of existence (object existence probability) as well as the probability that they are indeed friends (link probability existence) in order to provide a relevant measure of average age. When all friends' existence is certain, the expression will evaluate to their average age (if the friends are 15, 20 and 60 years, the average is 31.67 years). If one friend's existence probability decreases, her age's influence on the average will shrink proportionally. Figure 10 displays the probability distribution in the case where the three friends all have a 75 % existence probability.

The second category of related work comprises query and constraint languages such as SQL (Melton and Simon 1993) and OCL (Object Constraint Language 2010). Similarly to P^{2}AMF, these languages allow logical and arithmetic queries of object or entity models. They are, however, deterministic rather than probabilistic. There are however also probabilistic versions of such query languages, such as

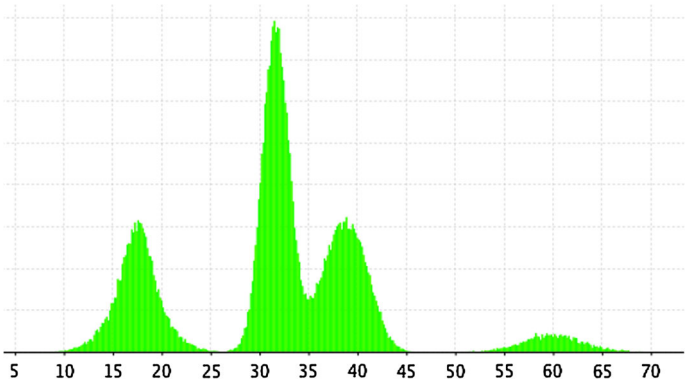

Fig. 10 Average age of three friends

PSQL (Dey and Sarkar 1998). These approaches have similarities with the second type of uncertainty introduced in Sect. 2.1, the existence of objects, but do not cover the stochastic attributes.

The third and most important category of related work is work on stochastic quality prediction for software architecture. Some of these, such as MARTE (UML Profile for MARTE 2009), KLAPER (Grassi et al. 2008) and ArgoPerformance (Distefano et al. 2005), are concerned with the analysis of UML or other Meta-Object Facility (MOF) compliant models. Others, such as the Palladio component model for model-driven performance prediction (Becker et al. 2009), the work by Meedeniya et al. (2012) on architecture based reliability evaluation and the work of the Q-ImPrESS EU project on performance, reliability, and maintainability (Becker et al. 2008) have opted for non-UML modeling formalisms. In the specific context of cloud computing, Stantchev proposes a method for performance evaluation (Stantchev 2009), Klems et al. (2009) offer a framework for economic value analysis, and Lee et al. (2009) propose metrics for non-functional Software-as-a-Service properties. However, common to all of these contributions is their focus on the analysis of particular properties. P^2AMF 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. (2012) on multiple non-functional property evaluation, using the Dempster-Shafer approach to probabilistic reasoning. However, P^2AMF 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 (Bass et al. 2003). These typically provide quite different support than P^2AMF, and are not based on probabilistic foundations.

8 Conclusions

It is desirable to predict and assess the expected quality and behavior of business and IT systems already in the design stage. Furthermore, in the constantly changing

complex and uncertain business environment of today, 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 business and IT system properties. The formalism, P^2AMF, supports automatic probabilistic reasoning based on set theory, first-order logic and arithmetics. Based on class and object diagrams, P^2AMF is compatible with UML. This paper has introduced P^2AMF and exemplified it for some simple analysis cases. The use of P^2AMF for predicting such diverse properties as system availability, interoperability, performance, usability, data accuracy and the effects of changes to the organizational structure of an enterprise has been reported on. Two algorithms for performing the required probabilistic inference was proposed, and a software tool supporting both modeling and inference was presented.

References

Aier S, Buckl S, Franke U, Gleichauf B, Johnson P, Närman P, Schweda CM, Ullberg J (2009) A survival analysis of application life spans based on enterprise architecture models. In: Proceedings of 3rd international workshop on enterprise modelling and information systems architectures (EMISA 2009), Lecture notes in informatics, pp 141–154

Allen RJ (1997) A formal approach to software architecture. PhD thesis, Carnegie Mellon University, Pittsburgh, PA

Altman DG, Gore SM, Gardner MJ, Pocock SJ (1983) Statistical guidelines for contributors to medical journals. Br Med J 286:1489–1493

Bass L, Clements P, Kazman R (2003) Software architecture in practice, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Reading, MA

Becker S, Trifu M, Reussner R (2008) Towards supporting evolution of service-oriented architectures through quality impact prediction. In: 23rd IEEE/ACM international conference on automated software engineering-workshops, 2008. ASE Workshops 2008, IEEE, pp 77–81

Becker S, Koziolek H, Reussner R (2009) The palladio component model for model-driven performance prediction. J Syst Softw 82(1):3–22

Briand L, Labiche Y, O'Sullivan L (2003) Impact analysis and change management of uml models. In: Proceedings of the international conference on software maintenance, IEEE, pp 256–265

Buschle M, Johnson P, Shahzad K (2013) The enterprise architecture analysis tool–support for the predictive, probabilistic architecture modeling framework. In: Proceedings of the 19th Americas conference on information systems to appear

Chen D, Doumeingts G (2003) European initiatives to develop interoperability of enterprise applications: basic concepts, framework and roadmap. Ann Rev Control 27(2):153–162

Claunch C (2011) Cloud computing can be the singular solution for at least five use cases. Techical report, Gartner

Cooke R, Goossens L (2004) Expert judgement elicitation for risk assessments of critical infrastructures. J Risk Res 7(6):643–656

Daly LE, Bourke GJ (2008) Interpretation and uses of medical statistics. Blackwell Science Ltd, Oxford

Dey D, Sarkar S (1998) Psql: a query language for probabilistic relational data. Data Knowl Eng 28(1):107–120

Distefano S, Paci D, Puliafito A, Scarpa M (2005) Design and implementation of a performance plug-in for the argouml tool. In: Proceedings of the 23rd IASTED international multi-conference on software engineering, IASTED

Drury C (2007) Management and cost accounting. South-Western

Eclipse Modeling Framework (2011) EMF: OCL plugin for the eclipse modeling framework. http://www.eclipse.org/emf/

Ferrer AJ, Hernández F, Tordsson J, Elmroth E, Ali-Eldin A, Zsigri C, Sirvent R, Guitart J, Badia RM, Djemame K, Ziegler W, Dimitrakos T, Nair SK, Kousiouris G, Konstanteli K, Varvarigou T, Hudzia B, Kipp A, Wesner S, Corrales M, Forgó N, Sharif T, Sheridan C (2012) Optimis: a holistic approach to cloud service provisioning. Future Gener Comput Syst 28(1):66–77. doi:10.1016/j.future.2011.05.022

Franke U (2012) Optimal IT service availability: shorter outages, or fewer? IEEE Trans Netw Serv Manag 9(1):22–33. doi:10.1109/TNSM.2011.110811.110122

Franke U, Buschle M, Österlind M (2013a) An experiment in SLA decision-making. In: Economics of grids, clouds, systems, and services. Springer, New York, pp 256–267

Franke U, Holm H, König J (2013b) The distribution of time to recovery of enterprise IT services (in review)

Franke U, Johnson P, König J (2013c) An architecture framework for enterprise IT service availability analysis. Softw Syst Model 1–29. doi:10.1007/s10270-012-0307-3

Freiling FC (2008) Introduction to security metrics. In: Dependability metrics. Springer, New York, pp 129–132

Friedman N, Getoor L, Koller D, Pfeffer A (1999) Learning probabilistic relational models. In: Proceedings of the 16th international joint conference on artificial intelligence, vol 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, pp 1300–1307

Gardner MJ, Altman DG (1986) Confidence intervals rather than p values: estimation rather than hypothesis testing. Br Med J (Clin Res ed) 292(6522):746–750

Gokhale SS (2007) Architecture-based software reliability analysis: overview and limitations. IEEE Trans Dependable Secure Comput 4(1):32–40

Gordijn J, Osterwalder A, Pigneur Y (2005) Comparing two business model ontologies for designing e-business models and value constellations. In: Proceedings of the 18th Bled eConference, Bled, Slovenia pp 6–8

Grassi V, Mirandola R, Sabetta A (2007) Filling the gap between design and performance/reliability models of component-based systems: a model-driven approach. J Syst Softw 80(4):528–558

Grassi V, Mirandola R, Randazzo E, Sabetta A (2008) 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, lecture notes in computer science, vol 5153, Springer, Heidelberg, pp 327–356. doi:10.1007/978-3-540-85289-6_13

Gustafsson P, Höök D, Franke U, Johnson P (2009) Modeling the IT impact on organizational structure. In: Proceedings of 13th IEEE international EDOC conference (EDOC 2009)

Halpern JY, Weissman V (2003) Using first-order logic to reason about policies. In: Proceedings of the IEEE computer security foundations workshop, IEEE computer society

Hansson H, Jonsson B (1994) A logic for reasoning about time and reliability. Formal Aspects Comput 6(5):512–535

Heath T, Martin R, Nguyen T (2002) Improving cluster availability using workstation validation. ACM SIGMETRICS Perform Eval Rev 30(1):217–227

Heckerman D, Meek C, Koller D (2004) Probabilistic models for relational data. Technical report, Microsoft

Holm H, Shahzad K, Buschle M, Ekstedt M (2013) P2cysemol: predictive, probabilistic cyber security modeling language (in review)

Jackson D (2002) Alloy: a lightweight object modelling notation. ACM Trans Softw Eng Methodol 11:256–290

Johnson P, Iacob ME, Välja M, van Sinderen M, Magnusson C, Ladhe T (2013) Business model risk analysis: predicting the probability of business network profitability. In: Enterprise interoperability. Springer, New York, pp 118–130

Joint A, Baker E, Eccles E (2009) Hey, you, get off of that cloud. Comput Law Secur Rev 25(3):270–274

Joseph M, Pandya P (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Jürjens J (2002) UMLsec: extending UML for secure systems development. Lect Note Comput Sci 2460:64–87

Khajeh-Hosseini A, Sommerville I, Sriram I (2010) Research challenges for enterprise cloud computing. Arxiv preprint arXiv:10013257

Klems M, Nimis J, Tai S (2009) Do clouds compute? A framework for estimating the value of cloud computing. In: Weinhardt C, Luckner S, Stößer J, Aalst W, Mylopoulos J, Rosemann M, Shaw MJ, Szyperski C (eds) Designing E-business systems. Markets, services, and networks, lecture notes in

business information processing, vol 22, Springer, Heidelberg, pp 110–123. doi:10.1007/978-3-642-01256-3_10
Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT Press, Cambridge, MA
Kurpjuweit S, Winter R (2007) Viewpoint-based meta model engineering. In: EMISA, vol 143, p 2007
Lee M, Kim H, Kim J, Lee J, Gum D (2005) StarUML 5.0 user guide
Lee JY, Lee JW, Cheun DW, Kim SD (2009) A quality model for evaluating software-as-a-service in cloud computing. In: 7th ACIS international conference on software engineering research, management and applications, 2009. SERA '09, pp 261 –266. doi:10.1109/SERA.2009.43
Lodderstedt T, Basin D, Doser J (2002) SecureUML: a UML-based modeling language for model-driven security. Lect Note Comput Sci 2460:426–441
Lyu MR (1996) Handbook of software reliability engineering. Mcgraw-Hill, New York
Marston S, Li Z, Bandyopadhyay S, Zhang J, Ghalsasi A (2011) Cloud computing-the business perspective. Decis Support Syst 51(1):176–189
Mason-Jones R, Towill DR (1999) Total cycle time compression and the agile supply chain. Int J Prod Econ 62(1–2):61–73
Meedeniya I, Aleti A, Grunske L (2012) Architecture-driven reliability optimization with uncertain model parameters. J Syst Softw 85(10):2340–2355
Melton J, Simon A (1993) Understanding the new SQL: a complete guide. Morgan Kaufmann Publishers, Los Altos, CA
Milch B, Marthi B, Russell S, Sontag D, Ong DL, Kolobov A (2005) Blog: probabilistic models with unknown objects. In: Proceedings of the 19th international joint conference on artificial intelligence, Morgan Kaufmann Publishers Inc.,Los Altos, CA, IJCAI'05, pp 1352–1359
Moriconi M, Qian X (1994) Correctness and composition of software architectures. SIGSOFT Softw Eng Notes 19:164–174
Närman P, Buschle M, Ekstedt M (2012) An enterprise architecture framework for multi-attribute information systems analysis. Softw Syst Model 1–32. doi:10.1007/s10270-012-0288-2
Nielsen J (1994) Usability engineering. Access Online via Elsevier
Object Constraint Language (2010) Object Management Group, version 2.3
OMG Unified Modeling Language (OMG UML) (2011) Superstructure. Object Management Group, version 2.4
Österlind M, Johnson P, Karnati K, Lagerström R, Välja M (2013) Enterprise architecture evaluation using utility theory. In: The trends in enterprise architecture research (TEAR) workshop
Osterwalder A, et al (2004) The business model ontology: a proposition in a design science approach. Institut dInformatique et Organisation Lausanne, Switzerland, University of Lausanne, Ecole des Hautes Etudes Commerciales HEC 173
Oxford Dictionaries (2013) Oxford Dictionaries Online. http://www.oxforddictionaries.com/definition/english/system
Puschner P, Koza C (1989) Calculating the maximum execution time of real-time programs. Real-Time Syst 1:159–176
Quatrani T (2002) Visual modeling with rational rose 2002 and UML, 3rd edn. Addison-Wesley Professional, Reading, MA
Randles M, Lamb D, Taleb-Bendiab A (2010) A comparative study into distributed load balancing algorithms for cloud computing. In: IEEE 24th international conference on advanced information networking and applications workshops (WAINA), 2010, IEEE, pp 551–556
Rappaport A (2000) Creating Shareholder value: a guide for managers and investors. Free Press, New York
Richards C (2004) Certain to win: the strategy of John Boyd, applied to business. Xlibris Corporation
Ritchey R, Ammann P (2000) Using model checking to analyze network vulnerabilities. In: Proceedings on IEEE symposium on security and privacy, 2000. S P 2000. IEEE, pp 156–165
Rold CD, Chamberlin T (2011) Cloud sourcing deals anatomy: from public to private, from services to technology lock-in. Technical report, Gartner
Ross JW, Weill P, Robertson DC (2006) Enterprise architecture as strategy: creating a foundation for business execution. Harvard Business Press, London
Schroeder B, Gibson G (2010) A large-scale study of failures in high-performance computing systems. IEEE Trans Dependable Secur Comput 7(4):337–350

Skene J, Emmerich W (2003) A model-driven approach to non-functional analysis of software architectures. In: Proceedings of the 18th IEEE international conference on automated software engineering, IEEE

Smith CU, Williams LG (2001) Performance solutions: a practical guide to creating responsive, scalable software. Addison-Wesley Professional, Reading, MA

Snow A, Weckman G (2007) What are the chances an availability SLA will be violated? In: Sixth international conference on networking, 2007. ICN'07, IEEE, pp 35–35

Sommestad T, Ekstedt M, Johnson P (2010) A probabilistic relational model for security risk analysis. Comput Secur 29(6):659–679

Spivey JM (1992) The Z notation: a reference manual. Prentice Hall International (UK) Ltd., Englewood Cliffs NJ

Stantchev V (2009) Performance evaluation of cloud computing offerings. In: Third international conference on advanced engineering computing and applications in sciences, 2009. ADVCOMP '09. pp 187 –192. doi:10.1109/ADVCOMP.2009.36

Steinberg D, Budinsky F, Merks E, Paternostro M (2008) EMF: eclipse modeling framework. Addison-Wesley Professional, Reading, MA

Ullberg J, Franke U, Buschle M, Johnson P (2010) A tool for interoperability analysis of enterprise architecture models using Pi-OCL. Enterprise Interoperability IV, pp 81–90

Ullberg J, Johnson P, Buschle M (2012) A language for interoperability modeling and prediction. Comput Ind 63(8):766–774

UML Profile for MARTE (2009) Modeling and analysis of real-time embedded systems. Object Management Group, version 1.0

van Sinderen M, Lagerström R, Ekstedt M, Johnson P (2012) Preparing the future internet for ad-hoc business networks support. In: Proceedingson 1st international workshop on architecture modeling for the future internet enabled enterprise (AMFInE)

Walsh B (2004) Markov chain Monte Carlo and Gibbs sampling