Complex Systems Informatics and Modeling Quarterly
CSIMQ, Article 64, Issue 11, June/July 2017, Pages 38–68
Published online by RTU Press, https://csimq-journals.rtu.lv
https://doi.org/10.7250/csimq.2017-11.03
ISSN: 2255-9922 online

Automated Probabilistic System Architecture Analysis in the Multi-Attribute Prediction Language (MAPL): Iteratively Developed using Multiple Case Studies

Robert Lagerström[1⋆], Pontus Johnson[1], Mathias Ekstedt[1],
Ulrik Franke[2], and Khurram Shahzad[1]

[1]KTH Royal Institute of Technology, Osquldas väg 12, 100 44 Stockholm, Sweden
[2]RISE Research Institutes of Sweden, SICS Swedish Institute of Computer Science, Kista Sweden

robertl@kth.se, pontusj@kth.se, mekstedt@kth.se, ulrik.franke@ri.se, khurrams@kth.se

Abstract. The Multi-Attribute Prediction Language (MAPL), an analysis metamodel for non-functional qualities of system architectures, is introduced. MAPL features automate analysis in five non-functional areas: service cost, service availability, data accuracy, application coupling, and application size. In addition, MAPL explicitly includes utility modeling to make trade-offs between the qualities. The article introduces how each of the five non-functional qualities are modeled and quantitatively analyzed based on the ArchiMate standard for enterprise architecture modeling and the previously published Predictive, Probabilistic Architecture Modeling Framework, building on the well-known UML and OCL formalisms. The main contribution of MAPL lies in the probabilistic use of multi-attribute utility theory for the trade-off analysis of the non-functional properties. Additionally, MAPL proposes novel model-based analyses of several non-functional attributes. We also report how MAPL has iteratively been developed using multiple case studies.

Keywords: System architecture, architecture analysis, system modeling, probabilistic analysis.

1 Introduction

Managing complex systems in modern enterprises is difficult. They have long life cycles and entail large investments [1]. There is also a growing perception among business leaders that their software system investments do not deliver the value promised [2]. This emphasizes the need to be able to predict the properties of future systems before they are commissioned.

⋆ Corresponding author

Reference: R. Lagerström, P. Johnson, M. Ekstedt, U. Franke, and K. Shahzad, "Automated Probabilistic System Architecture Analysis in the Multi-Attribute Prediction Language (MAPL): Iteratively Developed using Multiple Case Studies," Complex Systems Informatics and Modeling Quarterly, CSIMQ, Issue no. 11, pp. 38–68, 2017. Available: https://doi.org/10.7250/csimq.2017-11.03

Much of the model-based and -driven software engineering paradigm focuses on functional properties, i.e. ensuring that the software does what it is supposed to do. Questions such as; "Does the API accept the right parameters?", "Is data correctly pushed between model, view and controller?", "Are the files stored in the right place?" need to be answered. Indeed, these concerns are at the core of model-driven software engineering: when code is generated from a correct model, the code is also correct. Our focus, however, is different. Our aim is not to study functional properties, but non-functional ones.[1] Will the service be available 99.8% of the time? What will it cost? Can the data processed in the system be expected to be 99.5% correct? These questions are different from the functional questions, but no less important. Indeed, questions like these are key concerns for CIOs in their daily work [4].

The main original contribution of this article is the Multi-Attribute Prediction Language (MAPL). MAPL is a multi-attribute analysis metamodel for five non-functional qualities of system services, and a utility-calculation function that allows trade-offs between them. MAPL is probabilistic and provides fully automated analyses. While there is substantial literature on the analysis of non-functional qualities, the main contribution of MAPL lies in combining all of these analyses into a single unified analysis framework. Furthermore, for several attributes, the modeling and calculations are original contributions as well. Also, using a probabilistic framework in combination with architectural models is a novel way of addressing key issues in today's management of business and IT. Design and decision-making on an architectural level is inherently filled with uncertainty, which in turn calls for an approach that can cope with this state of affairs. A preliminary version of MAPL was described in teaching material used in a university graduate course [5] and a short version without any case studies was presented at a workshop for early feedback [6]. The teaching material also includes a number of video examples, that allow for a quick overview of the MAPL analysis capabilities.[2]

MAPL, and its predecessors, have been used in numerous case studies. Some of these have been reported earlier and others have not. In this article we describe the set of case studies that have been performed during the years 2006 to 2014. We report on two of these cases in more detail in order to show how MAPL was used.

The rest of the article is organized as follows: Section 2 contrasts our contribution with the existing literature. Section 3 briefly describes the modeling formalisms employed, before Section 4 explains the main contribution: the implemented prediction models for non-functional qualities. In Section 5 we report on the case studies performed using MAPL. Section 6 offers a discussion of the results, followed by Section 7 which concludes the article.

2 Related Work

One category of related work consists of contributions dealing with analysis of single non-functional qualities. These address quality analysis not explicitly connected to modeling, e.g. [7] on availability, [8] on data accuracy and [9] on software size measured in function points. These are more explicitly related to incorporating quality analysis into metamodeling, e.g. [10], [11] on availability, [12], [13] on data accuracy, and [14], [15] on application size and coupling. [16] offers an early example of quantitative analysis of enterprise architectures,

[1] The term non-functional property is increasingly being replaced or complemented with the term quality, as in the ISO/IEC 25010 standard [3]. According to this standard, system/software product quality consists of eight properties. In this article, we will use (non-functional) qualities and (non-functional) properties interchangeably.

[2] The interested reader can view the examples here: https://www.youtube.com/channel/UCWNzcSkeOi3hlGL9VRF4pVQ

adding queueing theory to the ArchiMate framework also used by us. However, these contributions all differ from ours in that they do not address multiple qualities at a time.

This leads to a second category of related work: previous modeling techniques that incorporate analysis of multiple quality attributes. For instance, the ABACUS tool [17] is able to analyze performance, reliability and total cost of ownership. This exhibits some overlap (cost and availability) with the MAPL qualities, but the ABACUS tool does not support analysis of accuracy, coupling, size or utility. In the area of software architecture, there are also a number of multi-attribute analysis methods. An early example is presented by Kazman et al. proposing a method for analyzing software architecture properties [18]. Gilmore et al. model performance, security and reliable messaging in service-oriented architectures [19]. Bengtsson and Bosch propose a method for reengineering software architectures, based on evaluation of reusability and maintainability [20]. However, these methods do not explicitly model trade-offs between the qualities assessed. In later work, Kazman et al. explicitly address the trade-off question, proposing the Architecture Tradeoff Analysis Method (ATAM) [21]. Thus ATAM is very similar in scope to MAPL, but focuses on performance, modifiability, and availability. The cost and accuracy assessments included in MAPL have no ATAM counterparts. Furthermore, ATAM is only partially automated. To deal with the lack of cost analysis in ATAM the Software Engineering Institute (SEI) developed the Cost Benefit Analysis Method (CBAM) [22] and later also an integration of these two [23]. Besides the differences in coverage and automation, ATAM and CBAM are methods that when employed could, for instance, benefit from the use of MAPL and its tool support. Furthermore, SEI has also released the Architecture Analysis and Design Language (AADL) [24] and for this tool support in the Open Source AADL Tool Environment (OSATE).[3] However, OSATE focuses on safety, security, and performance. Thus, there is little overlap in quality coverage.

Compared to most of the available methods and frameworks, like the ones discussed in the previous paragraph, our focus on automated probabilistic analysis is novel. The only somewhat similar work found would be Garlan et al. focusing on software engineering under uncertainty [25] and especially self-adaptation [26].

The most closely related work is probably the multi-attribute information systems analysis framework best described in [27]. The present contribution to some extent builds on that previous work, and this lineage is most evident in the probabilistic reasoning and the use of the ArchiMate metamodel. However, comparing the four qualities analyzed in [27] with the five qualities analyzed in this article, only availability and accuracy overlap. Furthermore, our use in this article of explicit utility modeling to make trade-offs between the qualities analyzed represents a significant improvement over the earlier work.

3 MAPL Background and Modeling Formalisms

The Enterprise Architecture Analysis Tool (EAAT) is a software for modeling and analysis of systems [28]. MAPL, the analysis mechanism, and EAAT have been developed for many years originating back to the paper "Enterprise Architecture Analysis with Extended Influence Diagrams" in 2007 [29]. In this paper the authors suggested that architecture models can and should be used for analysis of various properties (like security, availability, modifiability). The tool was made publicly available in September 2008. From the start, managing uncertainty has been a key feature of the EAAT tool, since this is important for analysis of system architectures.

[3] More information about OSATE can be found here: http://www.sei.cmu.edu/architecture/tools/analyze/index.cfm

The first formalism used was an extension of Influence Diagrams (themselves an extension of Bayesian networks). This evolved through Probabilistic Relational Models (PRMs) [30] into the Predictive, Probabilistic Architecture Modeling Framework (P^{2}AMF) now used [31].

3.1 P^{2}AMF

P^{2}AMF builds on the Unified Modeling Language (UML) and the Object Constraint Language (OCL), but adds a probabilistic inference mechanism.

From the perspective of a modeler, therefore, P^{2}AMF is very similar to UML and OCL. Indeed, every valid P^{2}AMF statement is also a syntactically valid OCL statement. However, the intended usage differs. OCL is used in the requirements phase to impose constraints on possible implementation alternatives expressed as UML diagrams. Inference in UML-OCL produces a list of OCL constraints evaluated as true or false, depending on whether the UML diagrams meet the constraints or not. While OCL is in this sense normative – imposing what *should* be the case – P^{2}AMF instead is predictive – describing what *will* (probably) be the case.

A key aspect of such prediction is managing *uncertainty*. Therefore, P^{2}AMF models include two kinds of uncertainty not found in standard UML models. First, *attribute values may be stochastic*. Whereas in a standard UML model, an attribute x will equal a specific value such as 4, in P^{2}AMF, the attribute may instead belong to a distribution, such as $Normal(4, 1)$, which may well be 4, but could also be 4.5 or 3.7.

Second, *the existence of objects and relationships may be uncertain*. Whereas in a standard UML model, a class A will either have a relation such as association with a class B, or not, in P^{2}AMF, there can be a probability of 40% that there is an association between the classes (and a corresponding 60% that there is not). Similarly, a probability can be assigned to whether there exists an instance of class A, or not. Formally, this probabilistic existence is introduced as a mandatory `existence` attribute on all classes and relationships.

From a modeler perspective, these are the only differences compared to UML-OCL: two mandatory `existence` attributes, and general attributes that are allowed to follow probability distributions. However, as seen in Section 4, this allows quite powerful modeling and prediction. In addition to these syntactic extensions to UML-OCL, the P^{2}AMF framework also consists of principles for how to make probabilistic inference in the models. The original publication describes in detail how two Monte Carlo approaches work in the P^{2}AMF context; rejection sampling and Metropolis-Hastings [31]. However, these details are not necessary to understand the contribution in this article.

As described above, P^{2}AMF is implemented in the EAAT tool. More precisely, there are two tools: an Object Modeler, which is used by the end user to model and analyze architectures and a Class Modeler, which is used to create class models, defining modeling and analysis principles. The MAPL class diagram depicted in Figure 1 is one such class model, that enables the end user to perform non-functional quality analysis of systems. To understand the division of labor between the two tools, note that the Class modeler is used to define which probability distributions or OCL expressions that govern attribute values and existence of classes or relationships, whereas in the Object modeler, these are instantiated, inference is carried out, and the resulting values are displayed as histograms. Figures 2, 3, 6, 7, 8, and 11 in Section 4 are all examples of object models.

A more detailed description of P^{2}AMF can be found in the original publication [31].

3.2 Evolution of Non-functional Quality Modeling

In our previous work, each non-functional quality was treated separately, resulting in numerous publications on, e.g. modifiability [15], security [32], availability [11], data accuracy

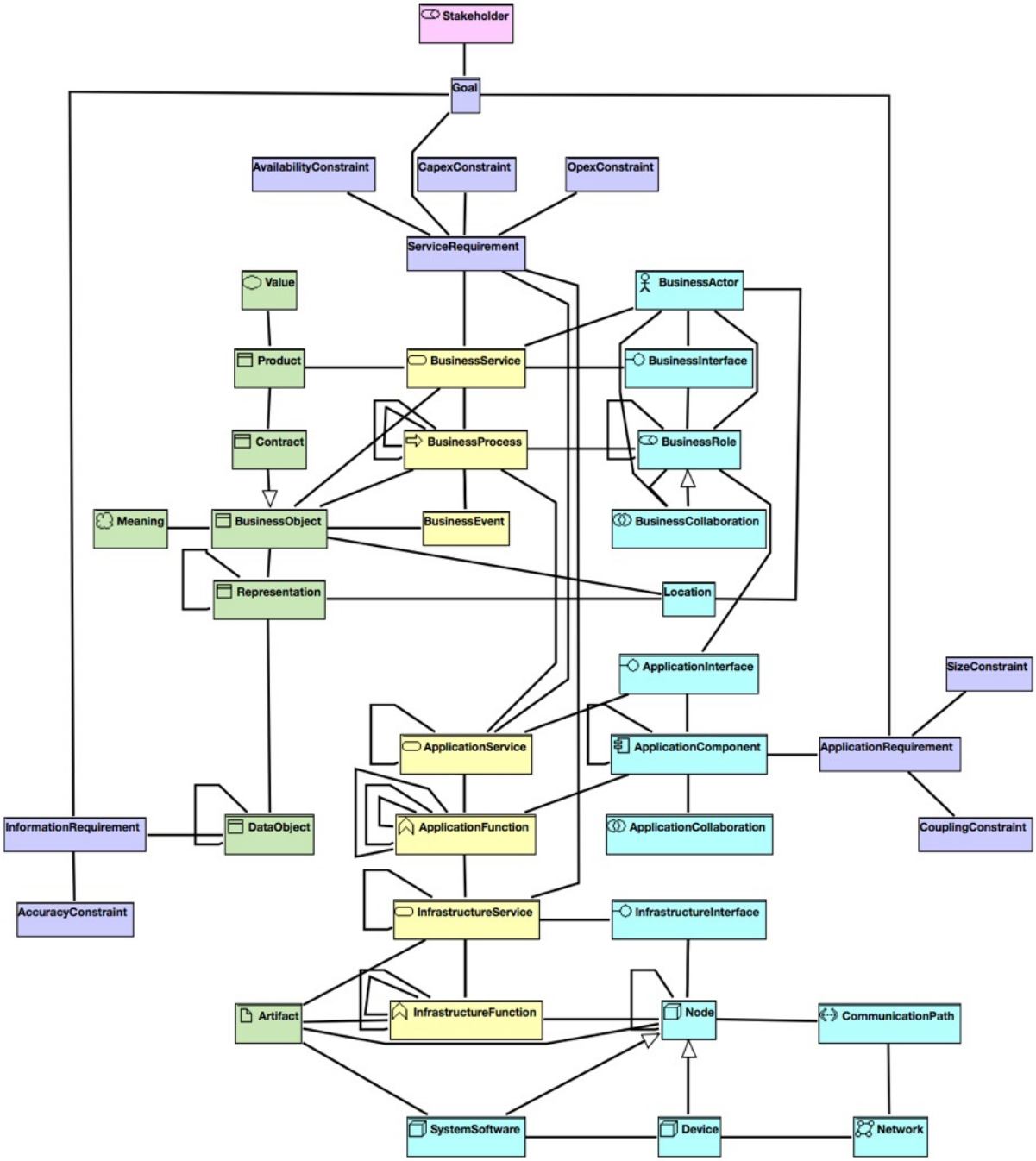


Figure 1. An overview of the classes and relations in MAPL. Attributes have been omitted for clarity.

[13], interoperability [33], and more. The first attempt at a unified framework was presented in [27], and the MAPL framework now presented in this article is the latest iteration. The evolution of MAPL and its modeling formalisms have been supported by numerous case studies, as reported in Section 5.

The MAPL class diagram in Figure 1 now also adheres to the ArchiMate standard for enterprise architecture modeling [34]. ArchiMate has gained popularity both among practitioners and in academia in recent years[4], and is used in the MAPL class diagram

[4] ArchiMate is, e.g. a part of The Open Group Architecture Framework (TOGAF) and its community: http://www.opengroup.org/subjectareas/enterprise/archimate

to enable a quick start for users already familiar with this modeling standard as well as future extensions.

Detailed descriptions of EAAT, P^{2}AMF, and ArchiMate are out of the scope for this article, but more information can be found in the original publications cited above.

4 Non-functional Quality Prediction

The main goal of the MAPL class diagram is to enable analysis of architecture models, leading to prediction of non-functional qualities. More precisely, the user can experiment with various potential to-be scenarios, trying out new architectural solutions, and testing the impacts of different numerical values of key attributes. This enables non-functional quality prediction in the sense of tool support for calculating ballpark figures on key concerns without having to actually implement a to-be architecture. In the following, we outline each of the qualities and describe their implementation. The small numeric examples illustrated in Figures 2-12 are fictitious minimum working examples, designed to convey how each quality is implemented. Real world case studies are discussed in Section 5.

Currently MAPL consists of five non-functional areas: service cost, service availability, data accuracy, application coupling, and application size. In addition, MAPL explicitly includes utility modeling to make trade-offs between the qualities. We have been elaborating with different qualities over the years, including, e.g. security, interoperability, and usability (as noted in Table 2). There are mainly two reasons why MAPL contains the current qualities; (1) during our case studies we have found these to be very important, and (2) there are fairly well agreed upon and well developed theories for how to analyze these. Other important aspects such as security and interoperability are both less well defined in terms of quantitative analysis and these typically require much more information in order to create a valid model, see [32] and [33].

As noted in both Section 2 and 5 each of the qualities have been developed and tested on their own before included in MAPL. Some of these have been published separately, but with slightly different focus, e.g. availability [11], modifiability (including, but not exclusively, size and coupling) [15], accuracy [13], cost [35], and utility [36].

4.1 Common Modeling Principles

The main classes in the metamodel, illustrated in Figure 1, are based on ArchiMate. Therefore, the three layers (i) Business, (ii) Application, and (iii) Technology/infrastructure are all used. Similarly, the three categories (i) passive (green in Figure 1), (ii) behavior (yellow in Figure 1), and (iii) active structure (blue in Figure 1) are also employed in the metamodel.

MAPL uses two kinds of attributes: *initial* attributes that are not influenced by other attributes, and *derived* ones that are calculated, at least partially, based on the values of other attributes. The user interacts with the model by setting *evidence* on the attributes. In MAPL, such evidence values of attributes should not be considered immutable facts, but rather play the role of probabilistic evidence[5]: an indication about what is the case, but not a final verdict. Most often, evidence are the only attribute values that a user needs to care about. Evidence can be deterministic – just a number, or stochastic – a full statistical

[5] In MAPL we adhere to the Bayesian school of statistics (rather than the frequentist school). We find it particularly suitable since it allows specifying the inference based on limited evidence data sets and incrementally update it with more data. The details of the inference logic as such, however, is not the focus of this article. Read more about Bayesian statistics, e.g. at: http://bayesian.org/Bayes-Explained

distribution. If evidence is not entered, attributes can have default values (specified in the class model). The default values built into MAPL are there to serve as demonstrations.

When working with MAPL, it is convenient to use the pre-defined views to address the different service qualities. These views also give a convenient overview of which classes are relevant for a particular analysis. In accordance to the concept of viewpoints used in ArchiMate, MAPL defines one viewpoint for each non-functional quality containing classes and relationships needed for the calculation of the quality.

In this article we have attached the OCL code for the availability analysis in order to give the reader a chance to understand the algorithms in our approach, see Appendix A. The complete code can be downloaded at the project website[6].

4.2 The Cost Viewpoint

Cost is a relatively straightforward system quality, but at the same time often a key concern for decision-makers. In MAPL, cost modeling is about tracking who uses various assets, in order to derive how costs can be shared across organizations. This can be used as input in, e.g. contractual discussions. We chose to employ a very traditional cost sharing approach, but we have also elaborated with other options in some case studies. For instance Activity Based Costing (ABC) could be a valuable add on for a future version of MAPL.

The costs to be shared are of two types:

- Capex (capital expenditure) is one-off costs. When customers purchase a computer, they pay for it once, and then own it.
- Opex (operational expenditure) is recurring costs. When someone hires a building for servers, the rent is paid, or when staff is hired for operations and maintenance, the salary is paid. [7]

In MAPL, the modeler sets costs on Active Structure Elements such as Business Roles, Application Components, and Nodes. In Figure 2, these cost-driving structure objects are the EmployeeDB and the Guard. The model then automatically derives the corresponding costs for Behavior Structure objects such as Business Process and Application Service. These are the business objects that share the resulting costs.

The principles for cost sharing are straightforward. If only one Behavior Structure object uses a particular Active Structure Element, then the Behavior Structure object carries the entire capex and opex of the Active Structure Element. In Figure 2, the PhysicalSecurity business process carries the entire cost of the Guard. If several Behavior Structure objects use the same Active Structure Element, they share the costs equally. In Figure 2, costs of the EmployeeDB are shared equally by the application services BuildingAccessControl and SalaryPaymentService. The total cost assigned to a Behavior Structure object is the sum of its fractions of the costs of all the Active Structure Elements it uses.

4.3 The Availability Viewpoint

When IT services become unavailable, they cease to provide the functionality intended and related business processes suffer. In MAPL, availability is about tracking how the availability of a service is affected by the availability of its constituent parts, i.e. components or services.

[6] MAPL downloads: https://www.kth.se/en/ees/omskolan/organisation/avdelningar/epe/research/resilient-informatio/projects/the-multi-attribute-prediction-map-class-diagram-1.387306

[7] The Opex attribute thus implies a time period, the cost is, e.g. per month. MAPL does, however, not explicitly defines this period. So when using MAPL this period must be decided by the end user and applied consistently over the whole model in order to make valid cost calculations.

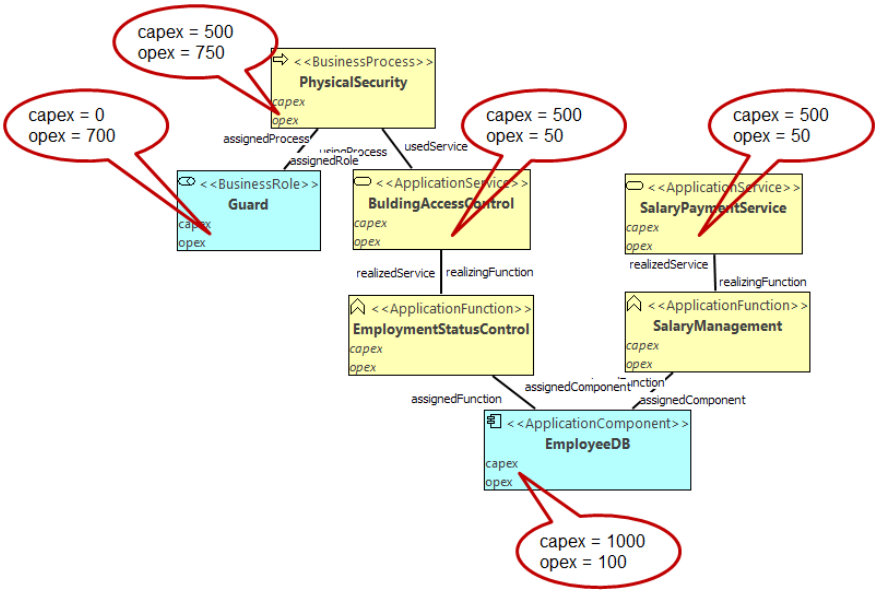


Figure 2. A simple MAPL cost model

In MAPL, availability can be modeled in two ways:

- As observed percentages – a service is available 97% of the time. This is expressed in the observedAvailability attribute, where the user can enter a percentage as evidence. In Figure 3, this attribute governs the availabilities of the VISA and PayPal application services.

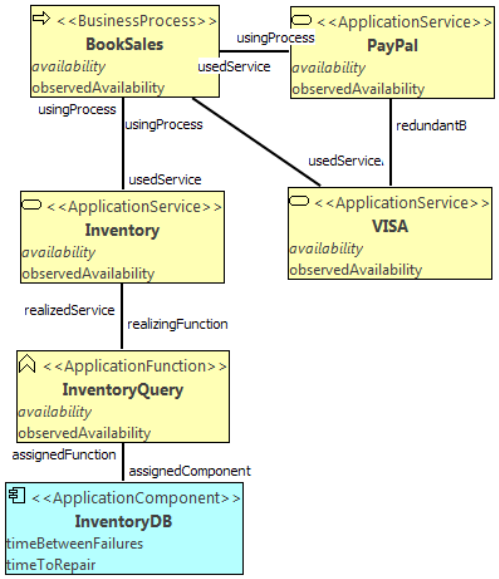


Figure 3. A simple MAPL availability model

- As a function of failure and repair rates – a service that fails often gets worse availability; a service that is quickly restored gets better availability. The availability of active structure elements is modeled using its timeToRepair (TTR) and timeBetweenFailures (TBF) attributes, and the resulting steady state availability is calculated according to Equation (1). In Figure 3, these attributes govern the availability of the InventoryDB application component.

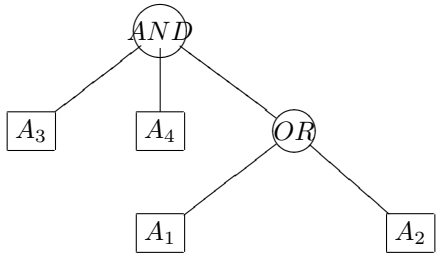
$$A = \frac{\text{TBF}}{\text{TBF} + \text{TTR}} \tag{1}$$

Note that Equation (1) is typically formulated using *mean values* (i.e. MTBF and MTTR). However, thanks to the probabilistic nature of P^{2}AMF, MAPL handles the full distributions. The user can enter evidence about full availability distributions in the TBF and TTR attributes. More precisely, the availability calculations use the subModel-functionality of P^{2}AMF, meaning that there are both an inner and an outer sampling loops as described in Figure 4.

```
Outer sampling loop:
  Inner sampling loop:
    TBF and TTR are sampled each by itself
    Availability is calculated as TBF/(TBF+TTR)
    isAvailable is sampled from bernoulli
        (TBF/(TBF+TTR)).
      Boolean; the system is up or down.
    isAvailable of depending objects is
        calculated by AND-OR-logic
        (accounting for redundancy).
  End of inner loop
  Availability attributes get a real value
    corresponding to the fraction of isAvailable-
    samples that were true in the inner loop.
End of outer loop
Availbility attributes are now distributions over
    the [0,1] interval of reals.
```

Figure 4. Availability sampling in MAPL

The AND-OR-logic of availability calculations is straightforward. When a behavior element depends on (uses, is realized by, is an assigned function of) other elements, its availability attribute will by default be calculated based on the availability of those other elements. This works recursively until the recursion bottoms out with the components at the bottom of the resulting fault tree. Typically, when an element depends on several elements, all of these have to be available for the dependent element to be available, i.e. AND-logic. The exception is the case of redundancy: the modeler can use a redundancy relation to specify that just one of the elements has to be available for the dependent element to be available, i.e. OR-logic. In Figure 3, the VISA and PayPal application services are redundant. An example of this recursion, with both AND and OR-logic, is illustrated in Figure 5.



$$A_{tot} = A_3 \wedge A_4 \wedge (A_1 \vee A_2)$$

Figure 5. A simple example of availability calculations. The isAvailable attribute of depending objects in the inner sample loop is the Boolean result of such a calculation.

4.4 The Data Accuracy Viewpoint

Data accuracy is important in all applications. Incorrect data leads to flawed decisions. Data accuracy addresses all the different ways in which recorded data deviates from the corresponding value in the real world, including deficiencies in correctness, precision, format, time lag, etc. In MAPL, data accuracy is about tracking how the accuracy of data changes as the data is processed throughout an architecture. To this end, MAPL models two kinds of changes in data accuracy:

- It can get more accurate, through correction. For instance, we check that all zip codes in a database have the right format.
- It can get less accurate, through deterioration. For instance, it can get old, or it can become distorted by applications.

Data accuracy is an attribute of Data Objects and Representations. In Figure 6, RawCustomerData is a representation, whereas SurveyData and ImprovedData are data objects. Deterioration and correction of data occurs in Behavior Structure objects such as Business Process and Application Service. In Figure 6, CustomerSurvey and the SurveyWithIncentives are the business processes that affect data quality, as governed by their correction and deterioration attributes. Changes in data quality can only happen as a result of behavior. The calculations are as follows: Data accuracy begins with an accuracy value that is directly specified in the attribute observedAccuracy. This always takes precedence over any derived attributes. When a Behavior Structure object reads data from one Data Object or Representation and writes data to another Data Object or Representation, the accuracy of the second Data Object depends on (i) the accuracy of the first Data Object, and (ii) how it is modified by the Behavior Structure. This is a three-part relation: Data1-Structure-Data2.

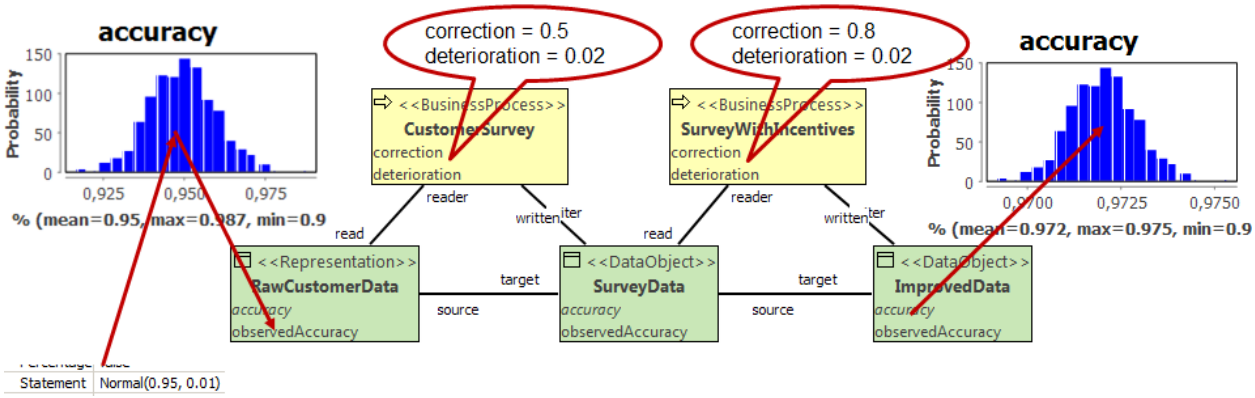


Figure 6. A simple MAPL accuracy model

Data accuracy can be calculated both in a deterministic and a stochastic fashion. When the original data accuracy is a distribution, as seen in the histogram to the left in Figure 6, all the resulting data accuracies throughout the chain also become distributions, as seen in the histogram to the right, illustrating the eventual accuracy of the data object ImprovedData.

4.5 The Application Coupling Viewpoint

An enterprise architecture consists of many different applications. Modeling application coupling is about keeping track of how they depend on each other. This makes it easier to plan and manage changes. Applications can be coupled in three ways:

- An application component can use services or functions of other application components (usage dependency).

- An application component can read or write to data objects that are read or written to by other application components (data dependency).
- An application component can share infrastructure resources with other application components (resource dependency).

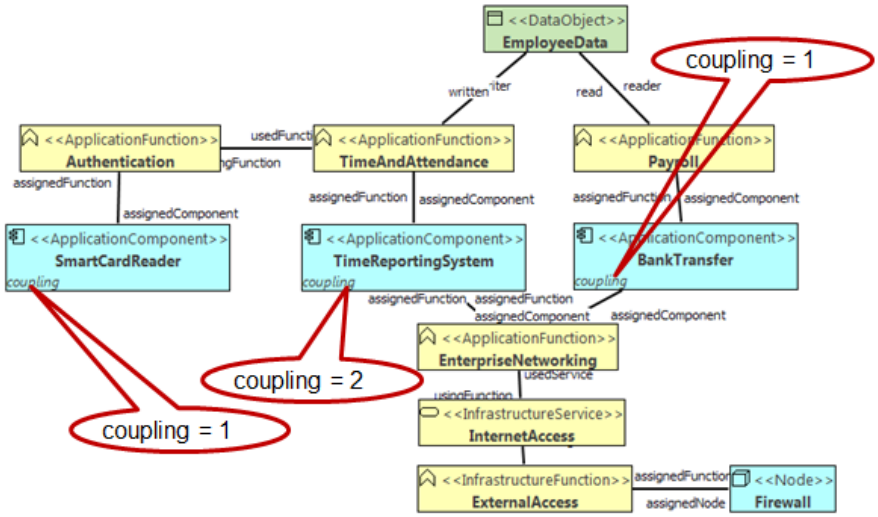


Figure 7. A simple MAPL coupling model

When two applications depend on each other in many ways, the coupling value does not increase. However, when applications depend on several other applications, their coupling values increase, regardless of the kind of dependency. Mathematically, the coupling dependencies are calculated using set operations. For any component, the coupling value is the cardinality of the union of three sets: its usage dependency set, its data dependency set, and its resource dependency set. In Figure 7, the TimeReportingSystem application component has a usage dependency set consisting of the BankTransfer application component, a data dependency set consisting of the same BankTransfer application component, and a resource dependency set consisting of the SmartCardReader application component. Thus, its dependency set consists of the BankTransfer and the SmartCardReader application components, resulting in a coupling value of two.

4.6 The Application Size Viewpoint

Knowing application sizes is useful when planning and managing changes. For instance, it is generally easier to modify small applications than large. Large applications are also more prone to contain errors, thus these often require more attention. Application size is reflected in the attribute linesOfCode. However, since lines of code is time consuming to keep track of for all applications in an enterprise we instead estimate the lines of code based on information already in the MAPL model through function points [37] and programming language used. The size is thus affected by several factors:

- How much data it reads or writes. If the application reads from a bigger number of data sources, its size grows. In Figure 8, the ParcelDisptach application component reads data from the two data objects InventoryData and CustomerData.
- How many business roles it interacts with. If the application interacts with more business roles through interfaces, its size also grows. In Figure 8, the ParcelDisptach application component interacts with the LoyaltyManager and ShippingDataManager business roles through interfaces.
- The programming language used.

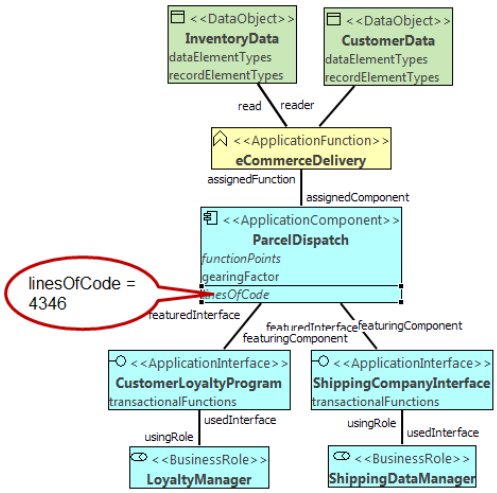


Figure 8. A simple MAPL size model

Mathematically, linesOfCode is calculated as the product of two other attributes: gearingFactor and functionPoints.

The function points analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user. Counting function points is done by considering the linear combination of five basic software components (inputs, outputs, master files, interfaces, and inquiries). In MAPL function points measure the size of the application in terms of data and transactions. The MAPL calculations are thus a simplified version of the well-known International Function Point User Group (IFPUG) standard [37], where more details can be found.

According to Quantitative Software Management (QSM), "the gearing factor is simply the average number of new and modified (Effective) Source Lines of Code (SLOC) per function point in the completed project. Gearing factors are calculated by dividing the effective code count for a completed project by the final function point count. SLOC counts represent logical, not physical line counts."[8] The gearing factor is language dependent and must be set by the modeler. Some gearing factor examples are given in Table 1.

Table 1. Gearing factors of a few different programming languages

Language	**Gearing factor**
C++	50
Java	53
SQL	21
Assembler	119

Functional size is measured in:

- The data objects written by the application components functions or services (called Internal Logic Files in the IFPUG standard).
- The data objects read by the application components functions or services (called External Interface Files in the IFPUG standard).
- The transactional functions as specified in the application interface (called External Input, External Output and External Inquiries in the IFPUG standard).

The functional size of a data object is based on the number of so called Data Element Types (DET) and the number of Record Element Types (RET). A DET is a unique, user

[8] See more at: http://www.qsm.com/resources/function-point-languages-table

recognizable, non-repeated attribute, such as "name", "year", "address", etc. A RET is a user recognizable sub-group of DET's, such as "employee". A data object may include a single or several RET's. The transactional function size is determined by the number of transactional functions in a user interface. (This is slightly simplified as compared to IFPUG.) A transactional function can be, for instance, (i) the update of a data object, (ii) some mathematical calculation, (iii) the sorting of a set of data, and more.

By calculating function points based on elements and interactions in the model instead of having a user input of the number of lines of code for each application saves time in the modeling phase, also it enables us to predict lines of code in to-be scenarios.

4.7 The Utility Viewpoint

"In an uncertain world the responsible decision maker must balance judgements about uncertainties with his or her preferences for possible consequences or outcomes. It's not easy to do and, even though we all have a lot of practice, we are not very good at it." This statement opens the book *Decisions with Multiple Objectives* [38] in which its authors present a formal technique helpful in decision making processes, hereby simple referred to as utility maximization.

A system architecture has a lot of different qualities (e.g. the Cost, Availability, Data Accuracy, Application Coupling, and Application Size). The notion of utility allows the modeler to make trade-offs between different qualities. This is a very important and valuable aspect in architecture modeling [39]. Including utility in architecture models allows us to keep track of:

- Whether the achieved quality of service fulfills the requirements set on a service.
- Which goals and stakeholders are affected by service requirements and their fulfillment.
- Trade-offs between different service qualities.

In MAPL, utilities are used on several levels. In a nutshell, the top-level *stakeholder utility* is derived from the lower level utilities, ultimately from whether and to what extent goals expressed as requirements on services have been achieved.

More precisely, *requirements utility* is a function of the weights and utilities of the connected constraints for each non-functional property. For example, a requirement on cost and availability translates into numerical constraints on these qualities (i.e. cost should be below some C, availability should be above some A) and into weights of their relative importances (i.e. meeting the cost constraint is given weight a, and meeting the availability constraint is given weight b). If the weights are calibrated to add up to unity, then requirements utility simply becomes the weighted mean of the constraint utilities. Formally, requirements utility is a multiplicative utility function as defined by [38], which degenerates into an additive utility function when the sum of all weights is one.

When the sum of constraint weights is much smaller than one, the multiplicative factors of the function dominate. These can be important in cases where one or several constraints singlehandedly may bring down the total utility to zero (this kind of weakest-link situation cannot be represented by the additive, weighted mean, function).

Goal utility is constructed in the same way, as a weighted function of the utilities of all requirements connected to the goal.

Finally, *stakeholder utility* is constructed in the same way, as a weighted function of the utilities of all goals connected to the stakeholder. If all goals are perfectly met, utility is one, while the total goal failure is represented by zero. If weights are calibrated to add up to unity, then stakeholder utility is the weighted mean of the goals' utilities.

The goal utility weight should equal the utility experienced by the stakeholder when goal achievement is at its best for this goal, but at its worst for all other goals. If the sum of

all weights equals 1, then the utility function is additive, i.e. a weighted mean function. If the sum of the weights is much smaller than 1, then the multiplicative components of the function dominate.

To summarize, stakeholder utility is a weighed function of goal utilities, which are in turn weighted functions of requirements utilities derived from constraint satisfaction.

The weighing of underlying utilities is done using the multiplicative utility function:

$$ku(x) + 1 = \prod_{i=1}^{n} [kk_i u_i(x_i) + 1] \tag{2}$$

In Equation (2) k_i is the weight for attribute i, x_i is the numeric representation of attribute i, x is the (vector) bundle of all numeric representations of attributes, $u(x_i)$ is the utility of attribute i, n is the number of attributes, and k is a scaling constant derived according to Equation (3):

$$1 + k = \prod_{i=1}^{n} (1 + kk_i). \tag{3}$$

Risk aversion. Whereas the utility attribute value is automatically given by the object that is subject to a requirement, the modeler must determine the value of *risk aversion*. The utility function of a risk averse stakeholder increases rapidly initially, and then tapers off. This is represented by a positive value of risk aversion. When risk aversion is zero, the utility curve is linear and the stakeholder is risk neutral. When risk aversion is negative, the stakeholder is risk seeking. The utility function, with risk aversion r, for a single attribute x (e.g. availability or cost) is given by:

$$u(r, x) = \frac{1 - e^{-r \frac{x - x_{\min}}{x_{\max} - x_{\min}}}}{1 - e^{-r}} \tag{4}$$

The interval $(x_{\min}, x_{\max})$ is the interval in which $0 < u(r, x) < 1$. Though Equation (4) is undefined for $r = 0$, it is straightforward to find its value in the limit as $r \to 0$, using l'Hôpital's rule:

$$\lim_{r \to 0} u(r, x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{5}$$

Figure 9 illustrates risk aversion using a few different r values, including the risk neutral $r = 0$, which is the straight line described by Equation (5).

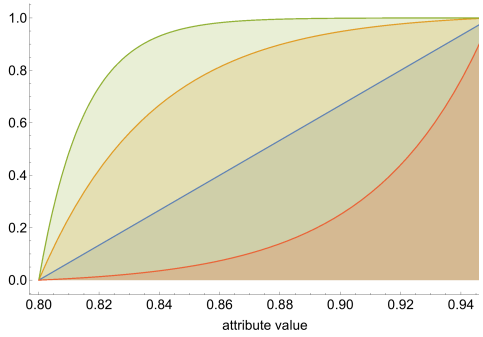


Figure 9. Risk aversion example, $r = \{0, 4, 10, -4\}$

Figure 10 illustrates the weighted goal utility for a risk averse stakeholder where the goal involves requirements on availability and accuracy.

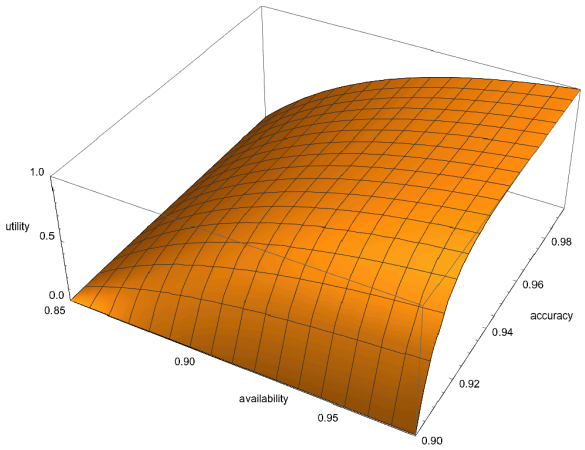


Figure 10. An example with multi-attribute utility and risk aversion. Parameters: $k_{avail} = 0.01, k_{acc} = 0.01, r_{avail} = 2, r_{acc} = 10$

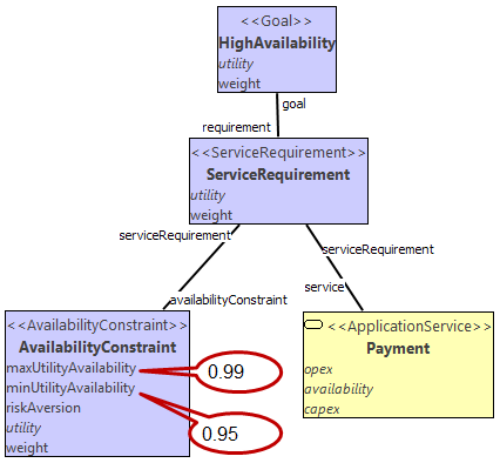


Figure 11. A simple MAPL utility model

Let us describe utility in relation to our MAPL entities. The basic modeling principle for utility is that Service entities stand in serviceRequirement relations to Requirement entities. In Figure 11, the Payment application service is related to a ServiceRequirement, which in turn is related to an AvailabilityConstraint. In general, the constraint entities are named for the qualities they represent. The constraint has attributes where the modeler can set minimum and maximum thresholds for how utility is gained from a quality. For instance, setting the minUtilityAvailability (denoted $x_{\min}$ above) attribute to 95% makes any lower availability worthless in terms of utility (whether it is 94% or 50%, it is simply too poor to be worth anything) and setting the maxUtilityAvailability (denoted $x_{\max}$ above) attribute to 99% correspondingly makes any higher availability not adding any more value (whether it is 99.01% or 99.99%, it is already so good that there is no more utility to be had). This principle is illustrated in Figure 12. In the probabilistic sampling, the values of the underlying qualities will sometimes generate no utility, sometimes maximum possible utility, and sometimes (perhaps most often) utility somewhere in between. Depending on the risk aversion, the curves will be different, as illustrated in the figure.

An important rationale for modeling and calculating with utility is to make trade-offs between several qualities. To make such calculations, more constraints can be added and set in relation to the relevant serviceRequirement. Each constraint (e.g. AvailabilityConstraint and OpexConstraint) have weight attributes, which enable their relative importance to be specified. By setting the appropriate weights, the modeler can capture precisely how

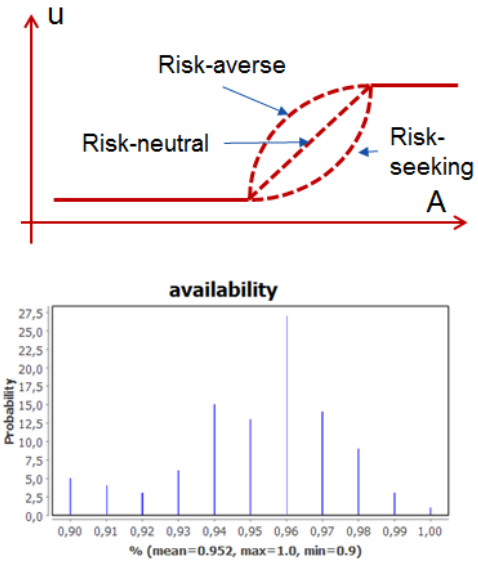


Figure 12. An illustration of how utility (sketch above) is derived from availability (histogram below) depending on the minUtilityAvailability and maxUtilityAvailability attribute

different qualities correspond to utility in his or her particular case. Additionally, weights can also be set on the serviceRequirement level, thus allowing for trade-offs between different requirements. Such aggregated utility is an attribute of Goal entities. More information about the use of utility theory in architecture analysis can be found in [36].

5 Case Studies

Our study builds on the foundations of design science [40], proposing a multi-attribute modeling language for analysis evaluated and iterated in multiple case-studies. Hevner et al. [40] suggest seven guidelines, which we have followed; 1) Design as an Artifact, 2) Problem Relevance, 3) Design Evaluation, 4) Research Contributions, 5) Research Rigor, 6) Design as a Search Process, and 7) Communication of Research.

1. **The Artifact.** The artifact produced is our proposed Multi-Attribute Prediction Language (MAPL).
2. **Problem Relevance.** Developing and managing complex systems is difficult. They have long life cycles and entail large investments. This makes it important to be able to predict various properties of future systems before deploying.
3. **Evaluation.** Each version of MAPL and its predecessors have been evaluated in one or more case studies, as outlined in the following parts of this section.
4. **Research Contributions.** In the end the main contribution is a unified framework for predicting multiple attributes including service cost, service availability, data accuracy, application coupling, and application size, as well as trading the importance of those attributes against each other.
5. **Research Rigor.** To achieve an easily adoptable artifact, we decided to adhere to UML [41] and OCL [42] through P^2AMF [31], as well as to ArchiMate [34].
6. **Search Process.** Our search process has been highly iterative and it included many cases with real systems and real problems to solve. We have done studies ourselves, we have collaborated with students, and we have had experts at the participating companies doing studies themselves.
7. **Communication of Research.** Early versions of the single attribute analysis frameworks and the analysis formalism have been published for the academic community.

The tool support has been demonstrated at academic conferences and at Swedish events for non-academics. Students in our courses have been handed a text book describing MAPL and its predecessors, accompanied by slides and videos.

In Section 3 we briefly described the history of MAPL. During all these years of iterative improvements of the framework, analysis formalism, and tool support we have conducted numerous case studies in terms of research projects reported in, e.g. [11], [15], [13], master thesis projects, master level education, and executive education.

Since the various research and master thesis project outcomes have been reported earlier we have in this article decided to focus on the case studies performed within our educational programs. Students studying computer science, electrical engineering, and industrial management can all decide to focus their final year on IT Management and more specifically our courses named IT Management with Enterprise Architecture I (EH2770) and IT Management with Enterprise Architecture II, Case Studies (EH2781).

In EH2770 students, typically in groups of four, have been using architecture modeling tools and metamodels for non-functional requirements analysis in parallel to our development of such frameworks, thus in 2007-2012 they used different versions of what we had to offer while 2013-2015 they all used MAPL and EAAT. In this course, the so called basic course on EA, they have applied the approach on fictive cases. In total 225 students have finished this course.

In EH2781 our students perform case studies with Swedish companies during a full semester (late August – early January). The companies provide the students with cases appropriate for architecture analysis. As in EH2770 the students used tools and frameworks in line with our current status in research, meaning MAPL and EAAT was first introduced in 2012. The last years we have also asked the students to extend MAPL based on the cases provided to them, i.e. extending our general framework to support more specific research questions. A complete list of cases can be found in Table 2. In total 132 students have successfully finished this course and together they have performed 46 case studies.

In parallel with the master education we have run a course for people in industry called Seminars in Industrial Information and Control Systems (EH221V), also known as Enterprise Architecture for Industry or Enterprise Architecture for Decision Making. The course started in 2009 and recent years the participants have used EAAT and MAPL on cases at their companies. We have had people attending from all business domains ranging from large multinational companies (like Ericsson and SAAB) through municipal and state agencies including the Swedish Armed Forces and Police to small consultancy firms. Examples of attending companies are; Alfa Laval, AMF, Banverket (Swedish Rail Administration), Arbetsförmedlingen (Swedish Public Employment Service), Biner Consulting, CIBER, Combitech, Ericsson, Folksam, GE Money Bank, Generic, H&M, Handelsbanken, ICA, Ilex Consulting, ITT, Jabberwocky, Klarna, Kronofogdemyndigheten (Swedish Enforcement Authority), Lantmäteriet (Public Authority for Surveying), Ln-4-solutions, Logica, NCC, Netlight Consulting, Nordea, Pravea, Procter & Gamble, SAAB Aerosystems, SBAB, Scania, SEB, SIX Financial Information, SJ (Swedish Railway), Skandia, Skatteverket (Swedish Tax Agency), Sollentuna Kommun (a local municipality), Stockholm University, Svenska Kraftnät (Swedish National Grid), Svenska Spel (Swedish State Gambling Company), Swedbank, Swedish Armed Forces, Swedish Police, Tele2, Telia, Teradata, Visab Consulting, and ÅF. In total 54 people have finished the course.

In this article we have selected two case studies performed by students in the course EH2781 IT Management with Enterprise Architecture II. The selected studies were run in 2012. We selected these two since they best fit the purpose of demonstrating the use of MAPL as described in this article. Due to our iterative work process of developing, testing, and improving MAPL over the years, not all the case studies are perfectly aligned with the

Table 2. Case studies performed 2006-2014 with master students in the course EH2781 IT Management with Enterprise Architecture II, Case Studies

Case Study Focus	Company (Year)
MAPL	AMF (2014[a], 2013[a], 2012[b]), Swedish State Gambling (2014)[b], SAAB (2013)[c], ABB (2013)[d], Swedish National Grid (2012)[d]
Multi-attribute	Gävle Energy (2009)[e], Vattenfall (2008)[f]
Interoperability	ABB FACTS (2011), ABB Ventyx (2011), Bombardier Transportation (2011), Sandvik (2008), E.On (2007), Ericsson (2006), Swedish Armed Forces (2006)
Modifiability	DSB (2010), Ericsson (2009), Handelsbanken (2009), ABB (2008), Swedish Motor-vehicle inspection (2007), Agria (2006)
Data quality	Volvo CE (2010), Hermelin Communication (2010), Swedish National Grid (2009), ABB (2007), Scania (2006)
Security	Jadestone (2010), Nacka Energy (2008), Fortum (2007), Scania (2006), Umeå Energy (2006)
Availability	Vattenfall (2010), Swedish Armed Forces (2007, 2006 x2)
Usability	Vattenfall (2008), Ericsson (2007), Fortum (2006)
Utility	Ortviken Paper Mill (2009), Swedbank (2008)
Performance	Fortum (2008), Swedish National Grid (2007)
Reliability	Swedish National Grid (2010)
IT-risk governance	ICA AB (2010)
Safety	Swedish Rail Administration (2007)

[a] Cost extensions.
[b] Flexibility extensions.
[c] Resource utilization extension.
[d] Response time extensions.
[e] Performance, availability, data quality, and security.
[f] Modifiability, performance, usability, and data quality.

modern MAPL version described in this article. Since we are presenting these cases as a part of our argumentation that MAPL works in practice it is important for us to show projects that are easy to understand and describe without the need to go into too much detail about extensions or new improvements.

5.1 Case 1: AMF

AMF is one of the leading pension companies in Sweden and also one of the largest owners on Nasdaq OMX Nordic Stockholm. They manage about 57 billion Euros in assets on behalf of their close to four million customers. The Swedish Trade Union Confederation and the Confederation of Swedish Enterprise jointly own AMF [9].

Basically, the aim of this study was to evaluate the IT landscape at AMF in terms of service availability, cost, modifiability, interoperability, data accuracy, and product flexibility. Based on an analysis of the as-is architecture, the goal was to suggest changes to the architecture (to-be scenarios) and based on the evaluation of these propose what improvements to make.

Data was collected through information available on the intranet at AMF, interviews with key personnel, and official system information available online. As delimitation in the study the focus was to highlight the Data Hub and the Service Hub, two critical systems that almost all other systems collaborate with (Figure 13). Also, the focus was to model the internet office, the business-to-business interface system, the customer relationship management system, the data warehouse, the master data system, and the printing system.

The main modifications tested in the to-be scenarios are related to removing dependencies between systems and increasing the data flows handled by the Data Hub and Service Hub in

[9] More information about AMF can be found here: www.amf.se

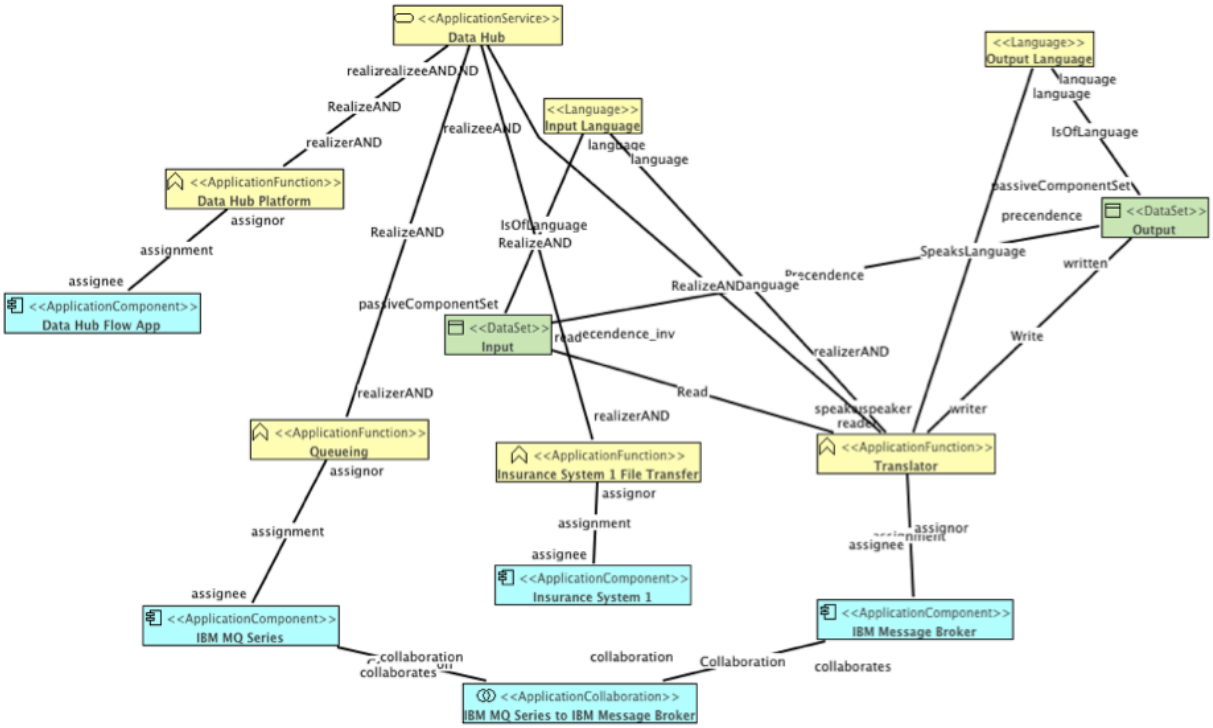


Figure 13. A view presenting the Data Hub application service at AMF

order to hopefully decrease complexity and increase performance. Also, one of the insurance systems is replaced with a new system with the aim to increase performance and modifiability.

By modeling the as-is and to-be scenarios including input data on, e.g. availability, cost, size, and accuracy (deterioration and correction) the students were able to help AMF to argue why certain changes are better than others. The main results showed that AMF could reduce complexity and improve availability for certain systems like the internet office and the CRM by investing more money in the Data and Service Hubs. However, the predicted return on the other suggested changes was not very substantial. As such, this is a perfect showcase for modeling as a means to prioritize among different possible actions, and select the ones most likely to succeed, without having to resort to trial-and-error.

5.2 Case 2: Swedish National Grid

Swedish National Grid (Svenska Kraftnät, short Svk) is the authority responsible for the power grid in Sweden. Their mission is to ensure that the power supply is reliable, environmentally friendly, and cost effective. One activity related to this mission is to monitor the electrical system around the clock. They also provide information about the grid to different stakeholders (customers)[10].

The aim of the Svk study was to suggest improvements of the customer relation process and supporting information systems. Svk typically has two types of customers, energy companies and electricity consumers. Both types interact with Svk through a web interface. Depending on their type, the customers can get different information via this interface, e.g. energy prices, where the electricity is generated, etc.

In the project, the current architecture related to the customer relation business service was modeled and possible future scenarios with appropriate modifications to the architecture

[10] More information about Svk can be found here: http://www.svk.se/en/about-us/

were analyzed. Data for the models was collected through interviews with, e.g. system owners, developers, and users. Also, for the availability analysis, logs were reviewed.

The first scenario revolves around an important application that produces output data used by the web interface for monitoring purposes, but also for billing specifications sent to the customers. This application also receives the data sent from the electric utility companies about their usage of the Svk grid. In the as-is situation, it is set up using four application servers, each of which offers a specific service. As a result, an outage on one server makes the corresponding service unavailable, and a high load on one server leads to poor performance, even though the overall server utilization is low. The solution evaluated in the first to-be scenario therefore introduces a load balancer for the application, leading to a dramatic increase in availability.

The second scenario addresses integration. In the as-is situation, integration between architecture components is achieved through FTP pushing and polling. A sending system puts a message file in an FTP directory. After a while, the integration platform notices the file and moves it to the inbox folder of the receiving system, which is continuously polled by the receiving system. This kind of integration entails long response times, and also offers poor usability, as users cannot get immediate feedback on malformed messages. The solution evaluated in the second scenario therefore is based on a service oriented architecture (SOA), where response times were reduced from a magnitude of minutes to a magnitude of milliseconds.

The third scenario addresses increased re-use and avoidance of duplicated functionality. In the as-is situation, two key systems both have their own separate database of company details such as contact information. The solution evaluated is to extend the web interface use and remove the corresponding functionality in the second system, resulting in cost savings.

In general, SvK had problems with their architecture mainly because of delaying refactoring and restructuring of the systems. When the web interface application was designed, there was an active decision in choice to use FTP push and polling due to the need of adaption to older systems. The general approach of delaying refactoring and the lack of building loosely coupled systems created a technical debt. The modeling and analysis in this case study helped them to realize the need to start working with these issues, define a future scenario, and plan how the company shall get there in time.

5.3 Case Summary

In our first case presented (AMF) the main problem addressed was to get a better understanding of why they have trouble adjusting to new regulations in a timely manner and with certain requirements fulfilled. Modeling and analyzing their current state with MAPL helped them collect and visualize information they had missed before. And the suggested scenarios for possible future directions were considered promising in order to increase flexibility without exceeding costs or worsening other important requirements.

"*To model a company's system landscape requires a lot of work and system landscapes are under constant change. I believe that there is a great potential in the approach since there is a lack of tools for analyzing architectures. Now, basically there are only tools to create an image of your current state, for instance, a system landscape or a company's business processes. Imagine a tool that in addition could, e.g. pinpoint solutions that defy principles and guidelines or suggest modifications in a design in order to follow a certain architectural style.*"

- Chief architect, AMF

In our second case (Svk) the main problem addressed was how to better utilize key customer and partner systems. For instance, one system was considered to have low availability and poor performance, another had long response times. Modeling and analyzing the current state with MAPL confirmed problems that they were experiencing and helped communicate these. The future scenarios propose both new solutions and some that have been discussed, but not implemented. With MAPL they got more information that could help them reach a decision about what way to improve their architecture.

"*It is often difficult to model your current architecture, not to mention modeling any future scenario. We have struggled to delimit ourselves in what to model in order to get the right support for making decisions. With the Multi-Attribute Prediction Language (MAPL) we get help to model the right things for important questions we need to answer. The built in analysis makes it easier to compare and create scenarios.*"

- Enterprise Architect, Svk

Although some solutions proposed in the case studies seem simple and straight forward, these are not always discussed, analyzed, and communicated in a good way at large corporations with many stakeholders and wills. With our approach test case study companies managed to pin down and show what the current architecture looks like and what issues they are experiencing with it. The future scenarios further aided the testing of different solutions and what effect these would have on the architecture and certain non-functional requirements.

6 Discussion

One of the benefits with MAPL is the combination of multiple attributes. If we model only one viewpoint, as suggested in some related works, it requires us to model a certain set of classes and relations. For instance, for the cost analysis viewpoint we model business roles, application components, and nodes in order to calculated costs for business processes, application services, and infrastructure services. Thus these classes need to be instantiated in an object model, which for most companies grow to be very large fairly quickly. If we model with MAPL, the structure can be utilized for analyses of other qualities. For instance, if we look at the object model for the cost viewpoint containing application components and application services we do not have to model these again for the availability viewpoint. Most of the MAPL qualities overlap with one or more classes. Thus, one of the benefits of using MAPL compared to a single analysis framework is the modeling effort synergy between the different viewpoints. Also, if an enterprise already has an ArchiMate model then it can be employed for analysis of MAPL attributes.

MAPL has successfully been used in teaching, both in a traditional graduate course on IT Management with Enterprise Architecture and in a practitioner oriented evening course, where the participants were all professionals working at the intersection of IT and business. This experience, and the feedback from the students, suggest that MAPL is both relevant and useful, though there is always room for improvement. Indeed, the development of the current version (MAPL 2.3) has been informed by this feedback. An example of one such improvement is the modeling of redundancy in the availability viewpoint. Whereas previous versions of MAPL employed AND and OR versions of the realization and usage relations, which turned out to be difficult to understand, the present version employs an explicit redundancy relation as described above in Section 4.3. Also it has become clear that the usability of the tool implementing MAPL is the key for further adoption in a wider community. In our case we have been using EAAT. This tool has been tailored to manage our specific interests of probabilistic analysis of architecture modelling, but when it comes

to general requirements such as usability, performance and collaborative working, it has to be improved considerably.

In the longer perspective, MAPL should be considered a first prototype step towards a larger and more comprehensive "CAD tool" for systems architecture analysis and simulation. Of course, each of the viewpoints described above is very simple, and a more comprehensive analysis would require adding a large number of additional attributes and viewpoints. However, MAPL offers a good structure for this as the explicit utility and requirements modeling enables trade-offs. Potential future candidates for inclusion in MAPL are a cyber security viewpoint based on, e.g. the CySeMoL language [32] and an interoperability viewpoint based on, e.g. the IF/ELSE language [33], which are both based on the same kind of probabilistic reasoning over models as MAPL.

Another MAPL dimension that could be improved, is to include more causal relations. The current language focuses largely on definitional relations between attribute values, e.g. the relation between timeToRepair, timeBetweenFailures and availability expressed in Equation (1). However, in many cases it would be even more interesting to analyze the *causes* of certain values, e.g. to conclude that the reason why it takes an hour to restore service after an outage is that there is only a single system administrator, and that the recovery time might be decreased if additional staff were hired. For a sketch of what such a causal theory in the availability case might look like, see [43, Section IV. C].

Additionally, an important direction for future work is to give the attributes used in MAPL standardized and unambiguous operationalizations. For instance, correct usage of the correction and deterioration attributes in the data accuracy viewpoint typically requires a number of important assumptions. Making such modeling practices clear to the user is, in the long run, a prerequisite for correct and reliable analysis.

This far our work has been focused on enterprise architecture or enterprise-wide IT architecture. However, as a probabilistic modeling and analysis approach the work could easily be extended to other types of complex systems, e.g. embedded systems and cloud infrastructure.

7 Conclusion

We have described the Multi-Attribute Prediction Language (MAPL); a comprehensive and integrated analysis framework for modeling and analysis (prediction) of non-functional qualities in both as-is and to-be architectures. The language includes the qualities; service cost, service availability, data accuracy, application coupling, and application size, as well as utility analysis for quality trade-offs. The main contribution of MAPL lies in combining all of these analyses into a single unified framework. Also, we report on numerous case studies performed during the evolvement of MAPL (2006-2014) and how these have been used in order to iteratively improve its modeling and analysis capabilities.

References

[1] S. Aier, S. Buckl, U. Franke, B. Gleichauf, P. Johnson, P. Närman, C. M. Schweda, and J. Ullberg, "A survival analysis of application life spans based on enterprise architecture models." in *EMISA*, 2009, pp. 141–154.

[2] C. Rettig, "The trouble with enterprise software," *MIT Sloan Management Review*, vol. 49, 2013.

[3] International Organization for Standardization, "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SquaRE) – System and software quality models," International Organization for Standardization, International standard ISO/IEC 25010:2011(E), March 2011.

[4] Å. Lindström, P. Johnson, E. Johansson, M. Ekstedt, and M. Simonsson, "A survey on CIO concerns – do enterprise architecture frameworks support them?" *Information Systems Frontiers*, vol. 8, no. 2, pp. 81–90, 2006. [Online]. Available: https://doi.org/10.1007/s10796-006-7972-0

[5] P. Johnson, R. Lagerström, M. Ekstedt, and M. Österlind, *IT Management with Enterprise Architecture*. Stockholm, Sweden: KTH Royal Institute of Technology, 2012.

[6] P. Johnson, R. Lagerström, M. Ekstedt, and U. Franke, "Modeling and analyzing systems-of-systems in the multi-attribute prediction language (MAPL)," in *Proceedings of the 4th International Workshop on Software Engineering for Systems-of-Systems*. ACM, 2016. [Online]. Available: https://doi.org/10.1145/2897829.2897830

[7] E. Marcus and H. Stern, *Blueprints for high availability, second edition*. Indianapolis, IN, USA: John Wiley & Sons, Inc., 2003.

[8] T. C. Redman and A. Blanton, *Data quality for the information age*. Artech House, Inc., 1997.

[9] J. E. Matson, B. E. Barrett, and J. M. Mellichamp, "Software development cost estimation using function points," *Software Engineering, IEEE Transactions on*, vol. 20, no. 4, pp. 275–287, 1994. [Online]. Available: https://doi.org/10.1109/32.277575

[10] A. Immonen, "A method for predicting reliability and availability at the architecture level," in *Software Product Lines*, T. Käköla and J. C. Duenas, Eds. Springer Berlin Heidelberg, pp. 373–422. [Online]. Available: https://doi.org/10.1109/32.277575

[11] U. Franke, P. Johnson, and J. König, "An architecture framework for enterprise IT service availability analysis," *Software & Systems Modeling*, vol. 13, no. 4, pp. 1417–1445, 2014. [Online]. Available: https://doi.org/10.1007/s10270-012-0307-3

[12] R. Y. Wang, M. Ziad, and Y. W. Lee, "Extending the ER Model to Represent Data Quality Requirements," in *Data Quality*, ser. Advances in Database Systems. Springer US, 2002, vol. 23, pp. 37–48. [Online]. Available: https://doi.org/10.1007/0-306-46987-1_3

[13] P. Närman, H. Holm, P. Johnson, J. König, M. Chenine, and M. Ekstedt, "Data accuracy assessment using enterprise architecture," *Enterprise Information Systems*, vol. 5, no. 1, pp. 37–58, 2011. [Online]. Available: https://doi.org/10.1080/17517575.2010.507878

[14] P. Fraternali, M. Tisi, and A. Bongio, "Automating function point analysis with model driven development," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*. IBM Corp., 2006, p. 18. [Online]. Available: https://doi.org/10.1145/1188966.1188990

[15] R. Lagerström, P. Johnson, and D. Höök, "Architecture analysis of enterprise systems modifiability–models, analysis, and validation," *Journal of Systems and Software*, vol. 83, no. 8, pp. 1387–1403, 2010. [Online]. Available: https://doi.org/10.1016/j.jss.2010.02.019

[16] M.-E. Iacob and H. Jonkers, "Quantitative analysis of enterprise architectures," in *Interoperability of Enterprise Software and Applications*, D. Konstantas, J.-P. Bourrières, M. Léonard, and N. Boudjlida, Eds. Springer London, 2006, pp. 239–252. [Online]. Available: https://doi.org/10.1007/1-84628-152-0_22

[17] K. Dunsire, T. O'Neill, M. Denford, and J. Leaney, "The ABACUS architectural approach to computer-based system and enterprise evolution," in *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*. IEEE, 2005, pp. 62–69. [Online]. Available: https://doi.org/10.1109/ecbs.2005.66

[18] R. Kazman, L. Bass, M. Webb, and G. Abowd, "SAAM: A method for analyzing the properties of software architectures," in *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press, 1994, pp. 81–90. [Online]. Available: https://doi.org/10.1109/ICSE.1994.296768

[19] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró, "Non-functional properties in the model-driven development of service-oriented systems," *Software & Systems Modeling*, vol. 10, no. 3, pp. 287–311, 2011. [Online]. Available: https://doi.org/10.1007/s10270-010-0155-y

[20] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 1998, pp. 308–317. [Online]. Available: https://doi.org/10.1109/icsr.1998.685756

[21] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," Carnegie-Mellon University, Tech. Rep., 2000.

[22] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001, pp. 297–306. [Online]. Available: https://doi.org/10.1109/icse.2001.919103

[23] R. L. Nord, M. R. Barbacci, P. Clements, R. Kazman, and M. Klein, "Integrating the architecture tradeoff analysis method (ATAM) with the cost benefit analysis method (CBAM)," DTIC Document, Tech. Rep., 2003.

[24] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," DTIC Document, Tech. Rep., 2006.

[25] D. Garlan, "Software engineering in an uncertain world," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 125–128. [Online]. Available: https://doi.org/10.1145/1882362.1882389

[26] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. ACM, 2006, pp. 2–8. [Online]. Available: https://doi.org/10.1145/1137677.1137679

[27] P. Närman, M. Buschle, and M. Ekstedt, "An enterprise architecture framework for multi-attribute information systems analysis," *Software and Systems Modeling*, vol. 13, no. 3, pp. 1085–1116, 2014. [Online]. Available: https://doi.org/10.1007/s10270-012-0288-2

[28] M. Buschle, "Tool support for enterprise architecture analysis : with application in cyber security," Ph.D. dissertation, KTH, Industrial Information and Control Systems, 2014.

[29] P. Johnson, R. Lagerström, P. Närman, and M. Simonsson, "Enterprise architecture analysis with extended influence diagrams," *Information Systems Frontiers*, vol. 9, no. 2-3, pp. 163–180, 2007. [Online]. Available: https://doi.org/10.1007/s10796-007-9030-y

[30] R. Lagerström, U. Franke, P. Johnson, and J. Ullberg, "A method for creating entreprise architecture metamodels: applied to systems modifiability," *International Journal of Computer Science and Applications*, vol. 6, no. 5, pp. 89–120, 2009.

[31] P. Johnson, J. Ullberg, M. Buschle, U. Franke, and K. Shahzad, "An architecture modeling framework for probabilistic prediction," *Information Systems and e-Business Management*, vol. 12, pp. 595–622, 2014. [Online]. Available: https://doi.org/10.1007/s10257-014-0241-8

[32] T. Sommestad, M. Ekstedt, and H. Holm, "The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures," *Systems Journal, IEEE*, vol. 7, no. 3, pp. 363–373, 2013. [Online]. Available: https://doi.org/10.1109/JSYST.2012.2221853

[33] J. Ullberg, P. Johnson, and M. Buschle, "A language for interoperability modeling and prediction," *Computers in Industry*, vol. 63, no. 8, pp. 766–774, 2012. [Online]. Available: https://doi.org/10.1016/j.compind.2012.08.009

[34] The Open Group, "ArchiMate 2.1 Specification," 2013, available: https://www2.opengroup.org/ogsys/catalog/C13L.

[35] P. Närman, T. Sommestad, S. Sandgren, and M. Ekstedt, "A framework for assessing the cost of it investments," in *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on.* IEEE, 2009, pp. 3154–3166. [Online]. Available: https://doi.org/10.1109/picmet.2009.5262271

[36] M. Österlind, P. Johnson, R. Lagerström, M. Välja *et al.*, "Enterprise architecture evaluation using utility theory," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International.* IEEE, 2013, pp. 347–351. [Online]. Available: https://doi.org/10.1109/edocw.2013.45

[37] IFPUG, "IFPUG Function Point Counting Practices Manual, Release 4.3.1," International Function Point Users Group, Tech. Rep., 2010.

[38] R. L. Keeney and H. Raiffa, *Decisions with multiple objectives: preferences and value trade-offs.* Cambridge university press, 1993. [Online]. Available: https://doi.org/10.1017/CBO9781139174084

[39] B. W. Boehm, "Value-based software engineering: Overview and agenda," in *Value-based software engineering.* Springer, 2006, pp. 3–14. [Online]. Available: https://doi.org/10.1007/3-540-29263-2_1

[40] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. pp. 75–105, 2004.

[41] Object Management Group. (2014) Unified Modeling Language (UML). [Online]. Available: http://www.omg.org/spec/UML/2.4.1/

[42] Object Management Group. (2014) Object Constraint Language (OCL). [Online]. Available: http://www.omg.org/spec/OCL/2.4/

[43] U. Franke, "Enterprise Architecture Analysis with Production Functions," in *IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC 2014).* IEEE, 2014, pp. 52–60. [Online]. Available: https://doi.org/10.1109/edoc.2014.17

Additional information about the article:

Author, ORCID iD	Robert Lagerström – www.orcid.org/0000-0003-3089-3885 Pontus Johnson – www.orcid.org/0000-0002-3293-1681 Mathias Ekstedt – www.orcid.org/0000-0003-3922-9606 Ulrik Franke – www.orcid.org/0000-0003-2017-7914 Khurram Shahzad – www.orcid.org/0000-0001-8433-6705
Article history	Received 18 April 2017 Revised 9 July 2017 Accepted 10 July 2017 Available online 31 July 2017
Article PII	S225599221700064X

A Appendix

MAPL code for availability analysis.

```
class BusinessProcess extends BehaviorElement
{
        operation affectedEntities() : Entity[*]
        {
                body:
                        usingBusinessProcess.oclAsType(Entity)->
                        union(realizedBusinessService.
                        oclAsType(Entity));
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        usedBusinessProcess.oclAsType(Entity)->
                        union(usedBusinessService.
                        oclAsType(Entity))->
                        union(usedApplicationService.
                        oclAsType(Entity))->
                        union(assignedRole.oclAsType(Entity));
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property realizedBusinessService#realizingBusinessProcess
        : BusinessService[*];
        property redundantA#redundantB : BusinessProcess[*];
        property redundantB#redundantA : BusinessProcess[*];
        property usedBusinessProcess#usingBusinessProcess
        : BusinessProcess[*];
        property usingBusinessProcess#usedBusinessProcess
        : BusinessProcess[*];
        property assignedRole#assignedProcess : BusinessRole[*];
        property usedBusinessService#usingBusinessProcess
        : BusinessService[*];
        property usedApplicationService#usingBusinessProcess
        : ApplicationService[*];
}

class ActiveStructureElement extends Entity
{
        operation isAvailable() : Boolean
        {
                body:
                        let availability : Real =
                        timeBetweenFailures/
                        (timeBetweenFailures+timeToRepair) in
                                bernoulli(availability);
        }
        attribute availability : ecore:
        :EDoubleObject { derived volatile }
        {
                derivation:
                        self.subModel(self.isAvailable())/100;
        }
        attribute timeBetweenFailures : ecore::EDoubleObject;
        attribute timeToRepair : ecore::EDoubleObject;
}

class InfrastructureFunction extends BehaviorElement
        {
        operation affectedEntities() : Entity[*]
        {
                body:
                        usingInfrastructureFunction.
                        oclAsType(Entity)->
                        union(realizedInfrastructureService.
                        oclAsType(Entity));
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        usedInfrastructureFunction.
```

```
                        oclAsType(Entity)->
                        union(usedInfrastructureService.
                        oclAsType(Entity))->union(assignedNode.
                        oclAsType(Entity));
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property realizedInfrastructureService#
        realizingInfrastructureFunction :
        InfrastructureService[*];
        property redundantA#redundantB :
        InfrastructureFunction[*];
        property redundantB#redundantA :
        InfrastructureFunction[*];
        property usedInfrastructureFunction#
        usingInfrastructureFunction :
        InfrastructureFunction[*];
        property usingInfrastructureFunction#
        usedInfrastructureFunction :
        InfrastructureFunction[*];
        property usedInfrastructureService#
        usingInfrastructureFunction :
        InfrastructureService[*];
        property assignedNode#assignedFunction :
        Node[*];
}

class BehaviorStructure extends Entity
        {
        operation isAvailable() : Boolean
        {
        body:
                if affectingEntities()->isEmpty() then
                        bernoulli(observedAvailability)
                else
                        areAffectingEntitiesAvailable
                        (affectingEntities())
                endif;
        }
        operation areAffectingEntitiesAvailable(aSet : Entity[*])
        : Boolean
        {
        body:
                let chosenSet : Set(Entity) = aSet->
                intersection(aSet->any(true)->closure(e:
                Entity | e.redundantSet())) in
                let remainingSet : Set(Entity) = aSet->
                asSet() - chosenSet in
                if remainingSet->notEmpty() then
                        chosenSet.isAvailable()->includes(true)
                        and areAffectingEntitiesAvailable
                        (remainingSet)
                else
                        chosenSet.isAvailable()->includes(true)
                endif;
        }
        attribute availability : ecore::EDoubleObject
        { derived volatile }
        {
                derivation:
                        if affectingEntities()->isEmpty() then
                                observedAvailability
                        else
                                self.subModel
                                (self.isAvailable())/100
                        endif
                        ;
        }
        attribute observedAvailability : ecore::EDoubleObject;
}

class Service extends BehaviorStructure
{
}
```

```
class ApplicationComponent extends ActiveStructureElement
{
        operation affectedEntities() : Entity[*]
        {
                body:
                        assignedFunction.oclAsType(Entity);
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property assignedFunction#assignedComponent :
        ApplicationFunction[*];
        property redundantA#redundantB : ApplicationComponent[*];
        property redundantB#redundantA : ApplicationComponent[*];
}

class BehaviorElement extends BehaviorStructure
{
}

class BusinessRole extends ActiveStructureElement
        {
        operation affectedEntities() : Entity[*]
        {
                body:
                        assignedProcess.oclAsType(Entity);
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property assignedProcess#assignedRole :
        BusinessProcess[*];
        property redundantA#redundantB : BusinessRole[*];
        property redundantB#redundantA : BusinessRole[*];
        property assignedActor#assignedRole : BusinessActor[*];
}

class ApplicationFunction extends BehaviorElement
{
operation affectedEntities() : Entity[*]
        {
                body:
                        usingApplicationFunction.
                        oclAsType(Entity)->
                        union(realizedApplicationService.
                        oclAsType(Entity));
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        usedApplicationFunction.oclAsType(Entity)
                        ->union(usedApplicationService.
                        oclAsType(Entity))->
                        union(usedInfrastructureService.
                        oclAsType(Entity))->
                        union(assignedComponent.
                        oclAsType(Entity));
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property assignedComponent#assignedFunction :
        ApplicationComponent[*];
        property realizedApplicationService#
        realizingApplicationFunction : ApplicationService[*];
        property redundantA#redundantB : ApplicationFunction[*];
        property redundantB#redundantA : ApplicationFunction[*];
        property usedApplicationFunction#
        usingApplicationFunction : ApplicationFunction[*];
        property usingApplicationFunction#
        usedApplicationFunction : ApplicationFunction[*];
```

```
        property usedApplicationService#
        usingApplicationFunction : ApplicationService[*];
        property usedInfrastructureService#
        usingApplicationFunction : InfrastructureService[*];
}

class Node extends ActiveStructureElement
{
        operation affectedEntities() : Entity[*]
        {
                body:
                        assignedFunction.oclAsType(Entity);
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property assignedFunction#assignedNode :
        InfrastructureFunction[*];
        property redundantA#redundantB : Node[*];
        property redundantB#redundantA : Node[*];
        property assignedArtifact#assignedNode : Artifact[*];
        }

        class BusinessService extends Service
        {
        operation affectedEntities() : Entity[*]
        {
                body:
                        usingBusinessProcess.oclAsType(Entity);
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        realizingBusinessProcess.
                        oclAsType(Entity);
        }
        property realizingBusinessProcess#
        realizedBusinessService : BusinessProcess[*];
        property usingBusinessProcess#
        usedBusinessService : BusinessProcess[*];
        property usingActor#usedService : BusinessActor[*];
        invariant
                cannotRealizeAndUseSimultaneously:
                self.realizingBusinessProcess.
                usedBusinessService->excludes(self);
}

class ApplicationService extends Service
        {
        operation affectedEntities() : Entity[*]
        {
                body:
                        usingApplicationFunction.
                        oclAsType(Entity)->
                        union(usingBusinessProcess.
                        oclAsType(Entity));
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        realizingApplicationFunction.
                        oclAsType(Entity);
        }
        operation redundantSet() : BehaviorStructure[*]
        {
                body:
                        redundantA->union(redundantB);
        }
        property realizingApplicationFunction#
        realizedApplicationService : ApplicationFunction[*];
        property usingApplicationFunction#
        usedApplicationService : ApplicationFunction[*];
        property usingBusinessProcess#
        usedApplicationService : BusinessProcess[*];
        property redundantA#redundantB : ApplicationService[*];
        property redundantB#redundantA : ApplicationService[*];
```

```
}

class InfrastructureService extends Service
{
        operation affectedEntities() : Entity[*]
        {
                body:
                        usingInfrastructureFunction.
                        oclAsType(Entity)->
                        union(usingApplicationFunction.
                        oclAsType(Entity));
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        realizingInfrastructureFunction.
                        oclAsType(Entity);
        }
        operation redundantSet() : BehaviorStructure[*]
        {
                body:
                redundantA->union(redundantB);
        }
        property realizingInfrastructureFunction#
        realizedInfrastructureService :
        InfrastructureFunction[*];
        property usingInfrastructureFunction#
        usedInfrastructureService : InfrastructureFunction[*];
        property usingApplicationFunction#
        usedInfrastructureService : ApplicationFunction[*];
        property redundantA#redundantB :
        InfrastructureService[*];
        property redundantB#redundantA :
        InfrastructureService[*];
}

class BusinessActor
{
        attribute ID : ecore::ELongObject;
        attribute name : String;
        property assignedRole#assignedActor : BusinessRole[*];
        property usedInterface#usingActor : BusinessInterface[*];
        property aggregatingCollaboration#aggregatedActor :
        BusinessCollaboration[*];
        property assignedLocation#assignedActor : Location[*];
        property usedService#usingActor : BusinessService[*];
}


class SystemSoftware extends Node
{
        property realizedArtifact#realizingSoftware :
        Artifact[*];
        property assignedDevice#assignedSoftware : Device[*];
}

class Device extends Node
{
        property assignedSoftware#assignedDevice :
        SystemSoftware[*];
        property network#device : Network[*];
}

class Entity
{
        operation isAvailable() : Boolean
        {
                body:
                        false;
        }
        operation redundantSet() : Entity[*]
        {
                body:
                        Set(Entity){};
        }
        operation affectedEntities() : Entity[*]
        {
                body:
```

```
                        Set(Entity){};
        }
        operation affectingEntities() : Entity[*]
        {
                body:
                        Set(Entity){};
        }
}
```