The Enterprise Architecture Analysis Tool – Support for the Predictive, Probabilistic Architecture Modeling Framework

Markus Buschle
Industrial Information and Control Systems
KTH Royal Institute of Technology
markusb@ics.kth.se

Pontus Johnson
Industrial Information and Control Systems
KTH Royal Institute of Technology
pontusj@ics.kth.se

Khurram Shahzad
Industrial Information and Control Systems
KTH Royal Institute of Technology
khurrams@ics.kth.se

ABSTRACT

The business of contemporary organizations is heavily dependent on information systems. Business processes and IT are interwoven and numerous technologies are in use. How the involved systems affect each other or impact the organizations' business domain is often uncertain, thus decision-making regarding information technology is challenging.

Enterprise architecture (EA) is a holistic, model-based management approach. Many of the available EA software tools focus on documenting and have limited analysis capabilities. In this article, a tool for EA analysis is presented, supporting the analysis of properties such as business fit, security, and interoperability. The tool is implemented to support the Predictive, Probabilistic Architecture Modeling Framework to specify and apply assessment frameworks for performing property analysis on EA models.

Keywords

Enterprise Architecture Analysis, probabilistic inference, system properties, software tool

INTRODUCTION

The management of organizations and their IT systems requires frequent decision-making. These decisions can, for example, be whether an existing system, in order to provide the performance needed for future services, should be replaced, upgraded or kept as it is. Another decision might be whether two applications should be integrated, used in the same fashion they have been used until now or whether the provided services should be outsourced.

In the making of machines, vehicles and buildings, the design technology CAD (Computer-aided Design) (Eastman 1999) is commonly used, i.e. models of the artifacts that will be created. These models provide a great benefit: it is easy to perform calculations of how the artifacts would behave instead of testing them empirically. Empirical testing such as crash tests is expensive and time consuming. Based on CAD calculations the right material for a given purpose can be identified or an optimal setup for a certain construction can be chosen.

One common approach for IT management, enterprise architecture (EA), is to create models of IT systems and the processes they support (Ross et al. 2006)(Lankhorst 2009). Similar to CAD tools, current EA tools support the creation of models representing organizations and their IT. However, the available EA tools generally only support the creation of descriptive models (Matthes et al. 2008) and lack advanced analysis capabilities. Investigations of system availability or how well an organization is capable of fulfilling its goals are generally not possible, neither is it possible to analyze cyber security aspects to identify vulnerabilities. Following the CAD analogy, current EA tools are comparable to CAD tools without the ability to analyze the design i.e. without the functionality to simulate crash tests, calculate stability of buildings or investigate the performance of engines.

Reports such as the Chaos report (The Standish Group International 2009) show that IT projects tend to expand in time or be more expensive than expected initially. This is often caused by unforeseen problems during the performance of the projects. Therefore EA tools capable of doing analysis would be useful to analyze the impact of changes already during the design stage and to help make the right decisions.

Within the process of decision-making, considering how different properties are related to each other and making trade-offs between those properties is often necessary (Närman et al. 2013). For instance, security is much improved by adding an intrusion detection system, but it may have a negative impact on performance.

Compared to mechanics of materials, the theories of EA are not as absolute, and thus, the uncertainties in these theories have to be taken into consideration in the analysis (Lagerström et al. 2009). Another challenging aspect is that collected information might be outdated, incomplete or even incorrect.

In this article, it is justified that there is a need for an EA tool to perform advanced analysis. This analysis is conducted based on the Predictive, Probabilistic Architecture Modeling Framework (P^2AMF) (Johnson et al. 2013), supporting the creation of models and the calculation of their properties. The framework allows considering the structure of the models and dependencies between modeled elements.

A tool that, compared to other approaches, has the following notable characteristics is presented[1]:

- It supports analysis of various analysis properties and is not limited to a predefined set of hardcoded analyses
- It considers dependencies between properties of the analyzed EA
- It handles incompleteness and uncertainty in the EA as well as in the theoretical foundations.

Considering the CAD analogy once again, the presented tool is a CAD tool for EA completely covering the use cases of conventional CAD tools. Models can not only be created and manually studied, but also analyzed. The tool helps identifying architecture scenarios that perform well with regard to considered analysis properties.

The rest of this article unfolds as follows: section 2 discusses related work. Next a method for EA analysis is described. In section 4 the underlying theory of the tool's calculation engine, P^2AMF, is explained. Section 5 describes the architecture of the presented tool. A dedicated section describing how the tool supports P^2AMF follows. Section 7 illustrates a case study applying the tool. Finally section 8 concludes the article and outlines future work.

[1] The tool can be downloaded at http://www.ics.kth.se/eaat

RELATED WORK

Several well-known EA frameworks exist. The perhaps most well-known one is the model taxonomy in the Zachman framework (Zachman 1987). Other frameworks focus more on a specific metamodel. Metamodels define the allowed content and graphical representations of the models. Some representatives are DoDAF (Department of Defense 2007) and ArchiMate (Lankhorst 2009). TOGAF (The Open Group 2008) emphasizes the method of EA and also features a metamodel. All those frameworks share a focus on the descriptive capabilities of EA (Kurpjuweit and Winter 2007).

Some of the most well-known EA tools include Rational System Architect (IBM 2013), ARIS IT Architect Designer (Software AG 2013), BiZZdesign Architect (BiZZdesign 2013), the Troux transformation platform (Troux Technologies 2013) and planningIT (alfabet AG 2013). These tools generally support one or several of the previously mentioned frameworks and metamodels.

Although the tools often possess some analysis capabilities, such analyses are qualitative, based on visualizations, and do not include calculations of model properties. E.g. it can be investigated which IT-systems support a selected business process, how many applications that read a certain data object or which roles are assigned to a specific department.

Within those tools EA models usually cannot be analyzed with respect to properties such as performance, business fit, availability or cyber security. Common EA tools allow relating entities, indicating that a certain concept e.g. is a specialization of another one, is composed of several concepts or is the predecessor of a modeled entity. Within the tools, relations visually indicate that the real concepts, which the model entities are meant to reflect, have a connection. A certain organizational role might be a specialization of another one, a business process might consist of several sub-processes, or an activity might be the predecessor of another one. However within the tools the relationships are not used to analyze the causal impact of properties of the modeled entities on each other. The tools do not infer the state of a considered attribute based on other attributes that are impacting it. Third, as well-known EA tools do not focus on advanced analysis capabilities, the tools do not cover the aspect of incomplete models. Those tools only consider elements that explicitly have been modeled. The fact that the tool user might not know all the details for creating a holistic model or might not be sure whether some aspects need to be described is not covered. Finally well-known tools expect the models to be correct in every detail. A user typically cannot express that he or she is unsure about the value of a certain attribute or relationship.

Abacus (Dunsire et al. 2005) however, offers complex analysis capabilities. It allows investigating EA models with respect to assessment properties such as performance, agility, and reliability. This is done using discrete-event and Monte-Carlo simulation. Abacus lacks two characteristics of the tool presented in this article. First, Abacus does not allow incorporating uncertainty in the analysis. Second, Abacus has a fixed set of analysis capabilities; the tool presented in this article allows the user to specify new or modify current analysis theories.

The tool providing the closest functionality compared to the one presented in this article is its own predecessor, the Enterprise Architecture Analysis Tool presented in (Buschle et al. 2010). This tool is a prototype with poor usability. It was implemented to support Probabilistic Relational Models (Koller 1999), a formalism with limited capabilities for considering structural aspects. Analyzing models under consideration of if and how objects are related is difficult to realize, as every possible combination of objects needs to be considered separately. The presented tool addresses this weakness by supporting P^2AMF, explained in section 4.

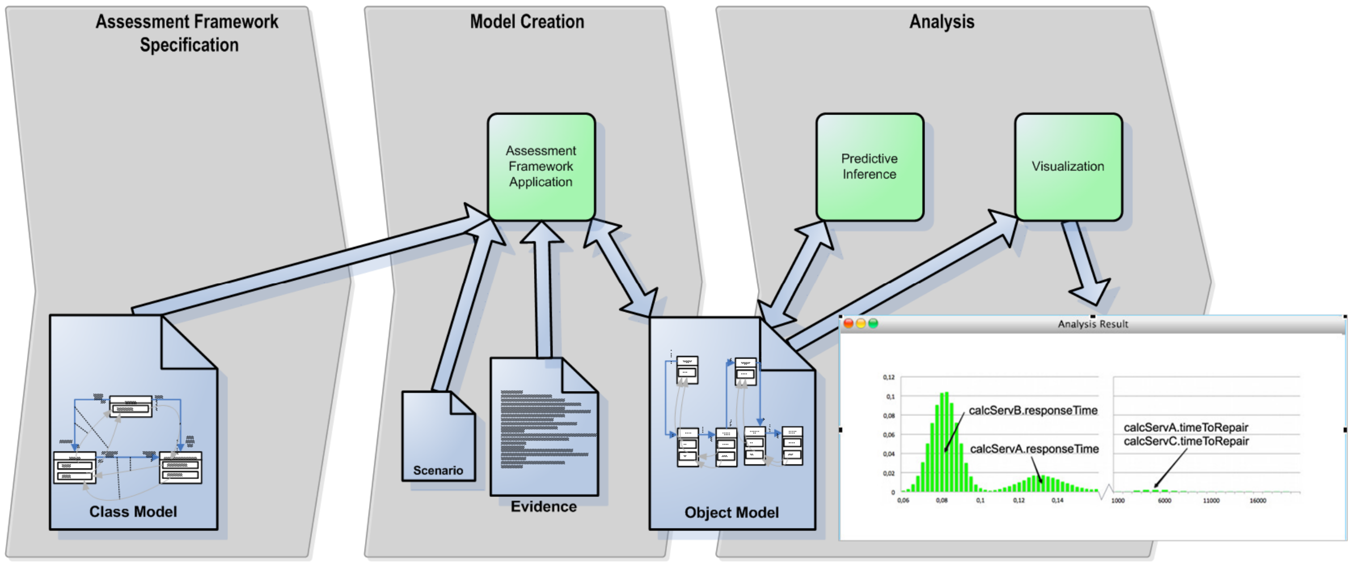


Figure 1 The supported enterprise architecture analysis method

ENTERPRISE ARCHITECTURE ANALYSIS

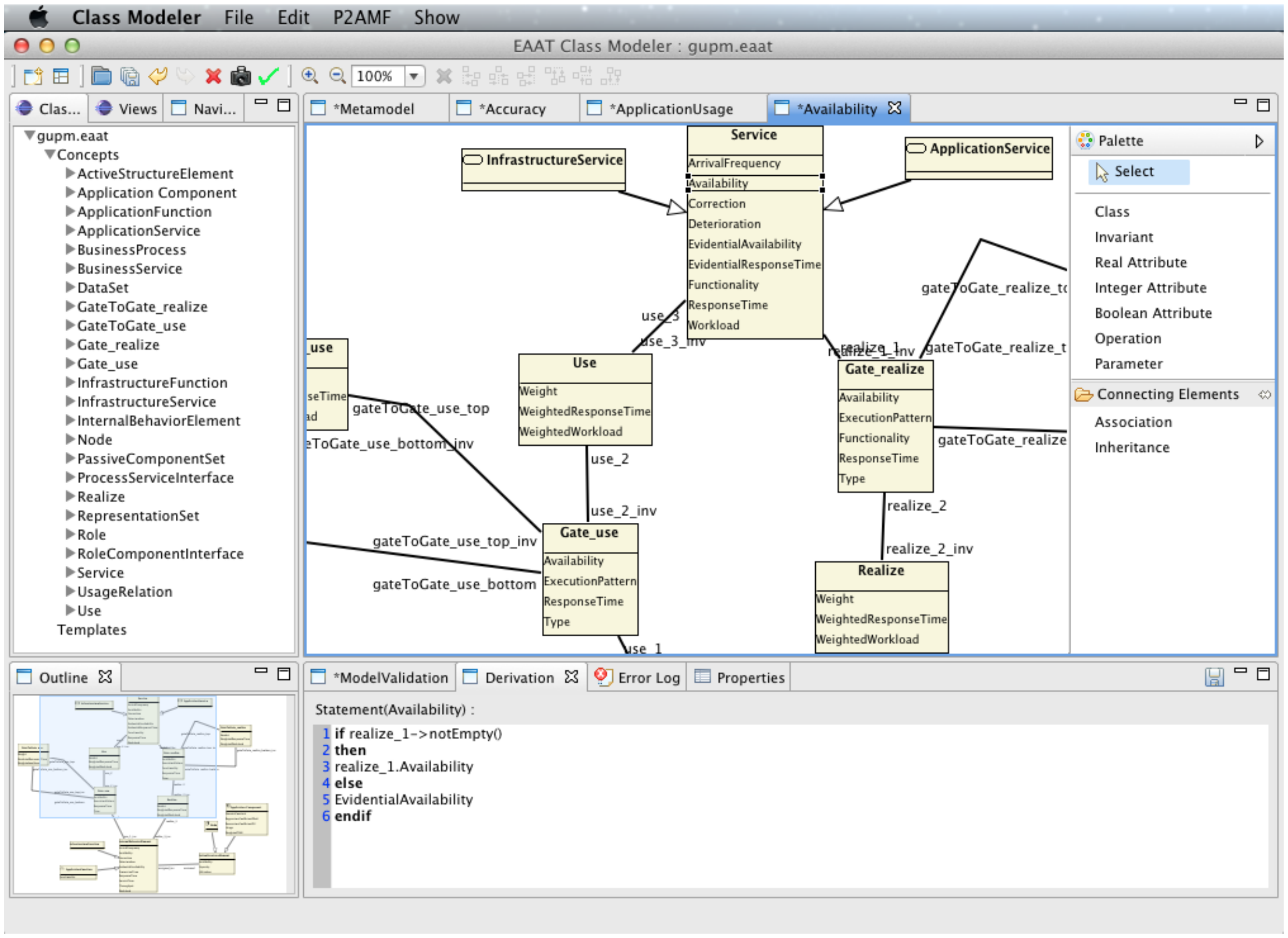
Several methods exist for performing analysis of enterprise architecture models for decision support. The presented tool supports the method presented in (Johnson and Ekstedt 2007). This method is depicted in Figure 1. First, an extended metamodel is created. Following the UML nomenclature, the authors refer to it as class model. This class model has an extended meaning. It describes not only the allowed content of the models, but also how characteristics of the model impact each other with regard to chosen criteria for architecture analysis. For those characteristics of the model general data describing them are included too. In the second step, scenarios of interest are identified. Each scenario is described as an object model instantiating the previously created class model. For a particular scenario one typically wants to replace general data with specific information, for some or all attributes that are part of the model. This is done in order to provide a more specific description. In the nomenclature of probabilistic inference, such instance-specific data is called evidence. In the final step, analysis, quantitative values of the models' quality attributes are inferred and the results are visualized.

The aim of this article is to address the challenge of creating a tool tailored specifically for EA analysis supporting decision-making. The primary use case for the tool, and the one portrayed in this article, is therefore to cover the method depicted and described above.

THE PREDICTIVE, PROBABILISTIC ARCHITECTURE MODELING FRAMEWORK

The Predictive, Probabilistic Architecture Modeling Framework (P^2AMF)(Johnson et al. 2013) is an extension of OCL (Object Management Group 2010) for probabilistic assessment and prediction of system properties. The main feature of P^2AMF is its ability to express uncertainties of objects, relations and attributes in UML-models and perform probabilistic assessments incorporating these uncertainties.

A typical usage of P^2AMF would be to create a model for predicting, e.g., the availability of an application. In P^2AMF, two kinds of uncertainty are introduced. First, attributes may be stochastic. When attributes are instantiated, their values are expressed as probability distributions. Second, the existence of objects and relationships may be uncertain. It may, be the case that one no longer knows whether a specific server is still in service. This is a case of object existence uncertainty. Such



```
Statement(Availability) :
if realize_1->notEmpty()
then
realize_1.Availability
else
EvidentialAvailability
endif
```

Figure 2 The Class Modeler

uncertainty is specified using an existence attribute *E* that is mandatory for all classes (here using the concept class in the regular object-oriented aspect of the word), where the probability distribution of the instance *myServer.E* might be:

P(*myServer.E*)=0.8

i.e. there is an 80% chance that *myServer* still exists. It might also be uncertain whether *myServer* is still serving a specific application, i.e. whether there is a connection between the server and the application. Similarly, relationship uncertainty is specified with an existence attribute *E* on the relationships.

The probabilistic aspects are considered in a Monte-Carlo fashion: For each iteration, the stochastic variables are instantiated with instance values according to their respective distribution. This includes the existence of classes and relationships, which are sometimes instantiated, sometimes not, depending on the distribution. Then, each of the P^{2}AMF statements is transformed into a proper OCL statement and can be evaluated. How this is realized in the tool is described below.

THE ENTERPRISE ARCHITECTURE ANALYSIS TOOL

This and the following section constitute the contribution of this article. They describe a tool implemented to support EA analysis for decision support. In order to realize this tool, the workflow of

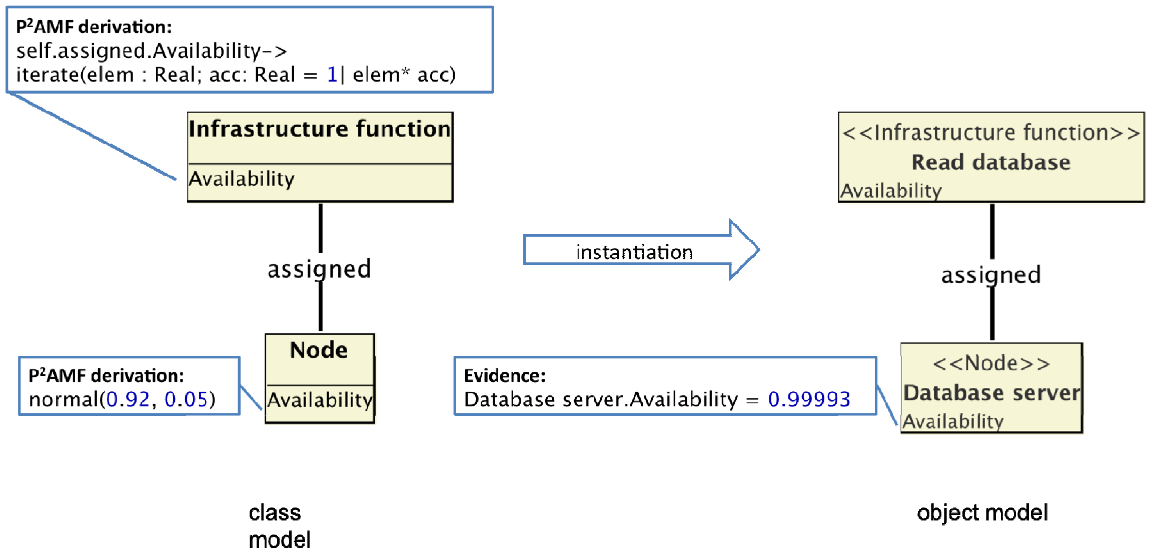


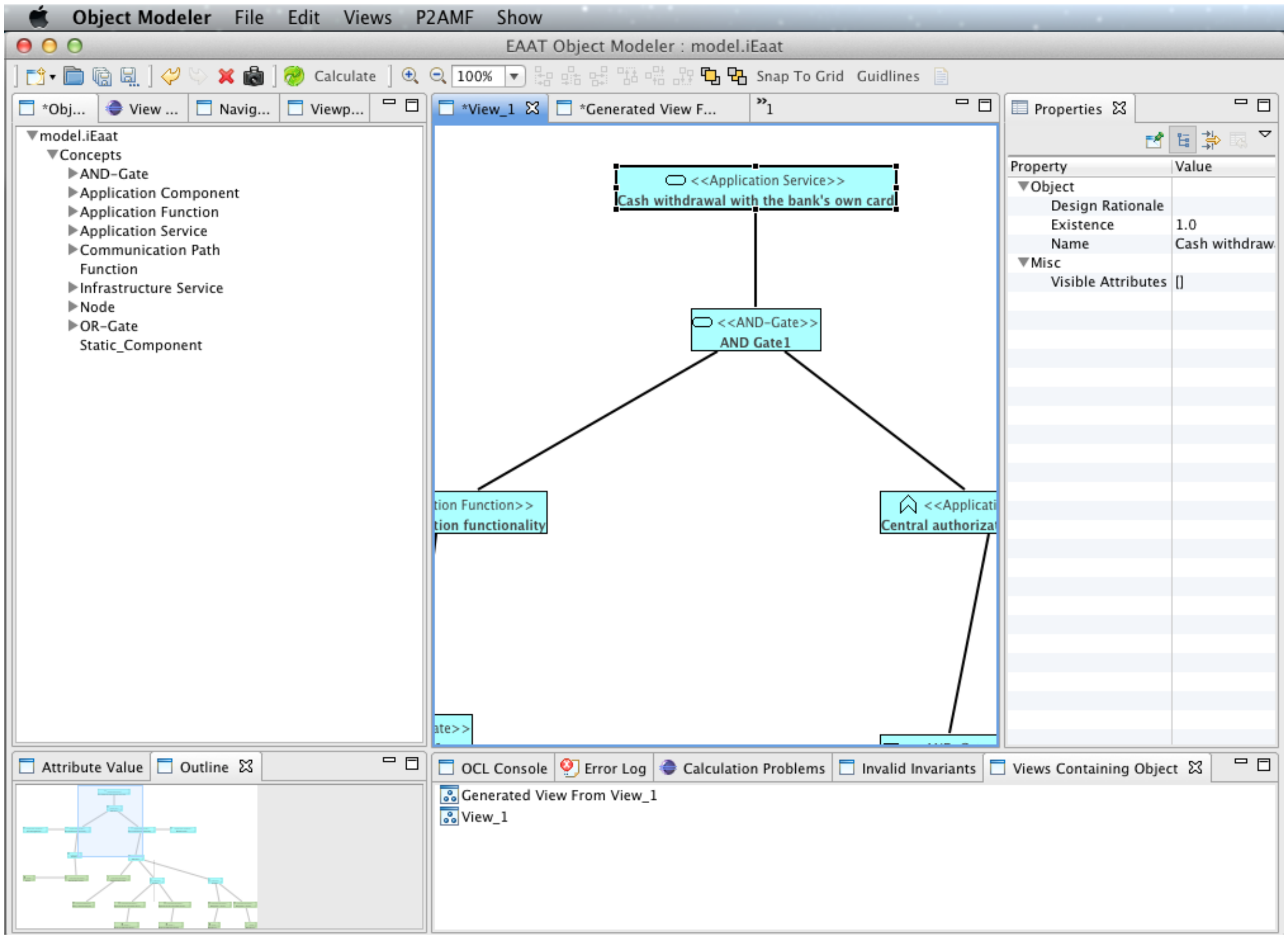
Figure 3 The described example

the previously described method for EA analysis is covered. Furthermore P^{2}AMF is supported to create class and object models and analyze the latter ones.

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 separated into two components, the Class Modeler and the Object Modeler, to be used in this order. The Class Modeler allows specifying an assessment framework (cf. the left part of Figure 1) in terms of classes, their attributes and the relations between them. Figure 2 contains a screenshot of the Class Modeler. A canvas for the creation of class models accounts for the largest part of the user interface. P^{2}AMF derivations can be specified in the lower part. To the right the defined classes with their attributes are shown and a graphical outline allows navigating through large models.

In case of an availability analysis the classes *Infrastructure function* and *Node* might be defined amongst others. They might be related to express that infrastructure functions typically are assigned to Nodes. The attribute *availability* might be added to both classes expressing that the criteria of analysis is availability. Thereafter a P^{2}AMF derivation might be specified expressing that the attribute *availability* of the *Infrastructure function* class depends on the availability of the assigned *Node* classes. For other attributes, such as the *Node.availability* attribute, initial probability distributions might be specified. Since these initial distributions are given on the class level, they represent the whole population of considered Nodes. Later, as the class model is instantiated, these estimates can be updated with instance specific data. The example is depicted in Figure 3. Class models are saved as ecore files using the EMF.

Once a framework has been specified it can be loaded into the Object Modeler. Figure 4 contains a screenshot of the Object Modeler. This component supports the application of the framework (cf. the middle part of Figure 1). The Object Modeler allows instantiating the classes that have been defined in the framework into objects and providing evidence for specific attribute values. By relating objects to each other a model of the scenario of interest can be created. Using the framework outlined in the previous paragraph a particular *Infrastructure function Read database* might be related to a specific *Node Database server*. For the D*atabase server* attribute *availability* evidence might be provided specifying that the considered server is known to have an availability of 99.993 % (cf. Figure 3). Once



Property	Value
Object	
Design Rationale	
Existence	1.0
Name	Cash withdraw
Misc	
Visible Attributes	[]

Figure 4 The Object Modeler

this model is in place, the Object Modeler allows calculating the attribute values included in the object model (cf. the right part of Figure 1). Thereby sampling according to P^{2}AMF is performed.

In the example the attribute *availability* of the *Read database Infrastructure function* is calculated based on the values provided for attribute *availability* of the *Database server*. How this exactly is done is the content of the following section. Once the sampling is completed the inferred attribute values are visualized (cf. again the right part of Figure 1).

The user interface looks similar to the one of the Class Modeler. Again, in the center a canvas for the creation of models, here object models, can be found. Properties of the modeled objects can be set using a tabular structure located to the right. Additional information regarding the created model or the performed analysis is visualized in the lower part. Finally the left part allows considering already modeled objects and navigating through the model.

SAMPLING ACCORDING TO THE P^{2}AMF

This section explains how inference is performed in the tool. The authors of P^{2}AMF (Johnson et al. 2013) are not specific on how this should be realized. In the tool three sampling algorithms, to infer the values of the attributes that are part of the created model, are implemented: forward sampling, rejection sampling and Metropolis-Hastings sampling, each having advantages and disadvantages.

As described above, the user starts the sampling functionality as soon as the object model describes the scenario of interest. The P^2AMF object model is sampled to create a set of deterministic object models. This is done considering the probability that a certain object is part of the created object model and that a given relationship is contained in that object model too. Therefore the existence attribute E, as explained above, is evaluated.

For each of these sample models, standard OCL inference is performed, thus generating sample values for all modeled attributes. For each attribute, the sample set collected from all sampled OCL models is used to characterize the posterior distribution.

For all sampling algorithms, the first step is to generate random samples from the existence attributes' probability distribution P(X): $x_1,\ldots,x_M$. For each sample, x_i, and based on the P^2AMF object model O^p, a reduced object model, $N_i \in N$, containing only those objects and links whose existence attributes, X_j, were assigned the value true, is created. Some object models generated in this manner will not conform to the constraints of 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. A set of traditional UML/OCL object models remains with $\Xi \subset N$, whose structures vary but are syntactically correct, and whose attributes are not yet assigned values. Finally, if the user provides evidence for one or several attributes, the sample is assigned the evidence value.

Forward sampling

Forward sampling (cf. Algorithm 1) consists of only a few steps and leads to a fast sampling process. However forward sampling comes with the disadvantage of not allowing the specification of evidence; on any arbitrary attribute in the object models, only evidence on attributes not calculated based on other attributes' values is allowed. In the example depicted in Figure 3 only evidence for the attribute *database server.availability* can be provided.

Forward sampling requires the attributes that are part of a sample Y_1, ..., 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. *Database server.availability* comes before *Read database.availability* in the example of Figure 3.

Following the general first step, as it was described above, the second step of the forward sampling algorithm is that for each of the remaining object models, Ξi, the probability distribution of the attributes not calculated based on the value(s) of other attributes, $P(Y^r)$ is sampled. This creates the sample set $y_1^r,\ldots,y_{\text{size}(\Xi)}^r$. 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 derivations are calculated in topological order for each remaining attribute in the object model, $y_j^{\bar{r}} = f_{y_j^{\bar{r}}}\ (Pa_{y_j^{\bar{r}}})$. The result is a set of deterministic UML/OCL object models, $\Lambda \subset \Xi$, where in each model, all attributes are assigned values. The final set of object models, $O \subset \Lambda$, contains attribute samples from the posterior probability distribution P(X,Y|e). These samples may thus be used to approximate the posterior.

```
for (int i=1; i<M; i++) {
        x=sampleExistenceAttributes();
        N = extractObjectModel(Op, x);
        if (syntacticallyCorrect(N)) {
                for(int j=1; j<N; j++) {
                $Y_j$ = sample(getParents($Y_j$));
                $\Lambda$ = assignAttributesToModel(y, N);
                O.add($\Lambda$);
        }
}
```

Algorithm 1 Forward sampling

Rejection sampling

The objective of rejection sampling (cf. Algorithm 2) is to generate samples from the posterior probability distribution P (X, Y|e), where $e = e^X \cup e^y$ denotes the evidence of existence attributes as well as the remaining attributes. The objective is thus to approximate the probability distributions of all attributes, given that 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.

Rejection sampling extends the previously described forward sampling algorithm with a third step. In this third step object models containing attributes not conforming to the evidence are rejected.

The sampling process ensures that root attributes always do conform, but this is not the case for OCL-defined attributes.

```
for(int i=1; i<M; i++) {
        x = sampleExistenceAttributes();
        N = extractObjectModel(Op, x);
        if (syntacticallyCorrect(N)) {
                y = sampleRemainingAttributes();
                $\Lambda$ = assignAttributesToModel(y, N);
                if (conformsToEvidence($\Lambda$)) {
                        O.add($\Lambda$);
                }
        }
}
```

Algorithm 2 Rejection Sampling

As described above, rejection sampling extends forward sampling. Doing so, it overcomes the weakness of only allowing the specification of evidence on the root attributes. The pseudo code above shows that this is implemented as a filter, where samples confirming to the evidence are kept and all others are rejected. This proceeding is costly, as it requires the creation of many samples in order to generate a sufficient number of valid samples.

Metropolis- Hastings sampling

Metropolis-Hastings sampling (Hastings 1970; Walsh 2004)(cf. Algorithm 3) is an iterative sampling technique converging to a desired distribution limit. It aims at creating 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.

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.

```
Randomly create $x_{init}$
MC.add($\Lambda_{init}$)
for(int i=1; i<M + B; i++) {
    $\Lambda'$ = generateNewSample($\Lambda$);
    $P(\Lambda'|\Lambda)$ = calculateProbability($\Lambda', \Lambda$);
    $\alpha$ = calculateProbabilityOfAcceptance($\Lambda', \Lambda, P(\Lambda'|\Lambda)$ );
    if ($\alpha$ < l){
        MC.add($\Lambda'$)
    }
    else{
        MC.add($\Lambda$)
    }
}
removeBurn-InSamples(B)
```

Algorithm 3 Metropolis-Hastings sampling

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 $P(x'|x)$ is evaluated. Thereafter the probability of acceptance α of the sample is calculated, considering the likelihood $P(x'|x)$, which over time is given more consideration. If α 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.

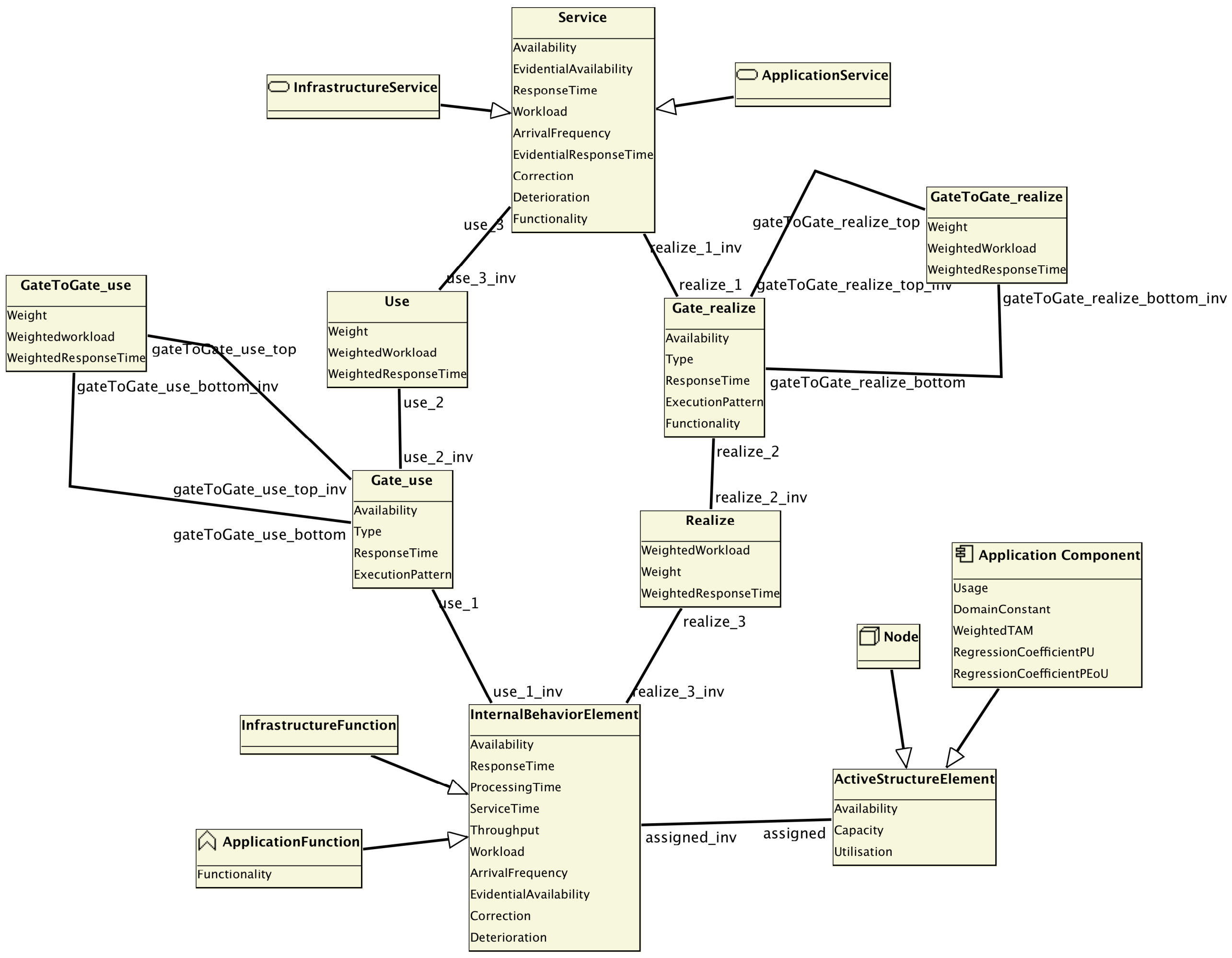


Figure 5 The used class model

Similar to rejection sampling, Metropolis-Hastings sampling allows specifying evidence for any attribute of the model. This algorithm does need a comparably fewer number of samples and is therefore, especially when considering models including a large number of attributes, more effective. The biggest disadvantage of Metropolis-Hastings sampling is that, especially for models with many local minima, the best solution might not be found. This is because of the chain structure of the result, where samples are based on their predecessor.

CASE STUDY

This section describes a case study, applying the tool in practice. A Nordic bank used the tool to analyze the availability of an application service provided for 10 million customers. A class model for availability analysis (Närman et al. 2013) was employed. The class model allows investigating availability at enterprises and is built upon ArchiMate (Lankhorst 2009). It makes use of the fault tree formalism. This class model (cf. Figure 5) was modeled in the Class Modeler. The class model employs the ArchiMate categories Active Structure Element and Behavior Element. It uses generalization i.e. introduces the *Function* class, not included in ArchiMate, in order to express that several concepts are

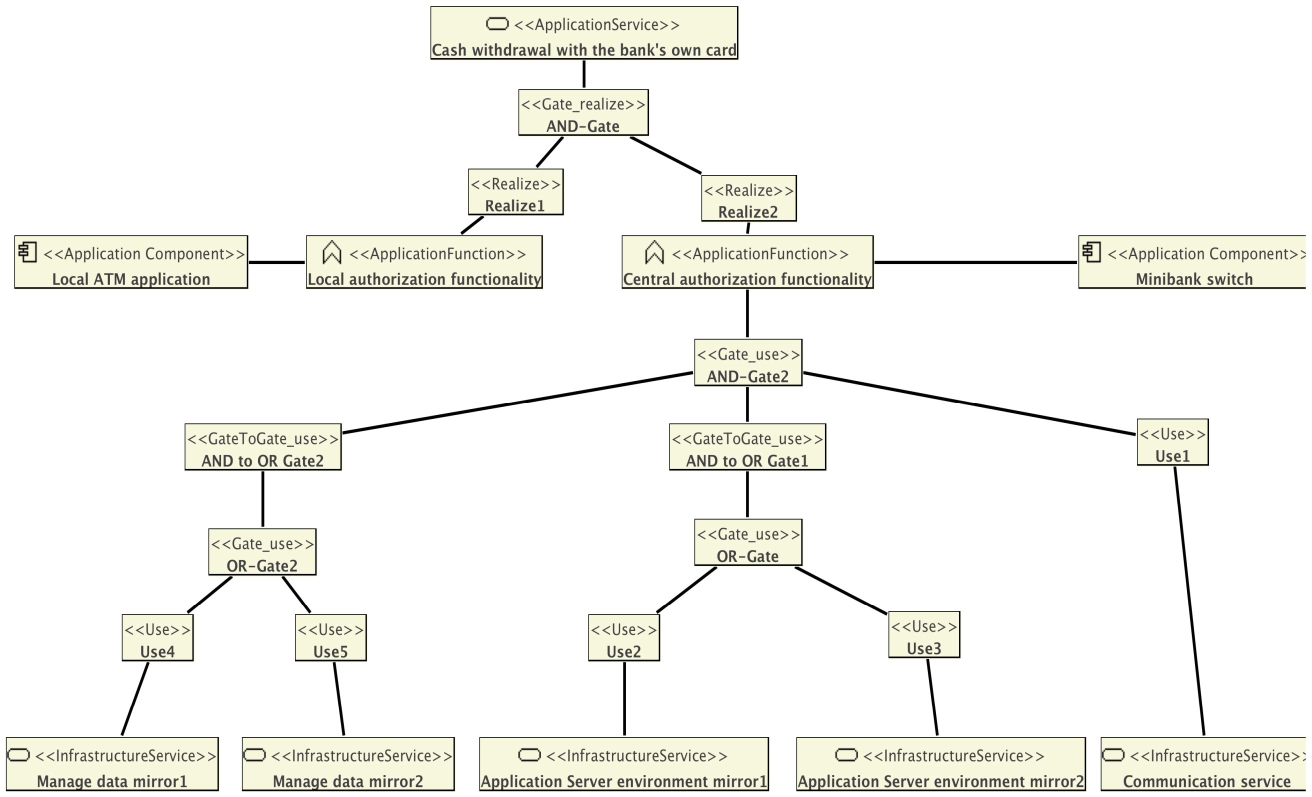


Figure 6 The resulting model (attributes not visualized)

alike from an availability perspective. For example the class *Active Structure Element* is used to generalize *Application Component* and *Node*. Furthermore the metamodel of ArchiMate was extended to express logical gates, as they can be found in fault trees. Logical gates depict relations between functions. *Gate_Use* can be used to describe that *Services* use *Internal Behavior Elements*; vice versa *Gate_Realize* allows describing that an *Internal Behavior Element* is realized by one or several services. The presented class model allows modeling And-Gates and OR-gates using the attribute *Type* of the Gate classes. AND-Gates describe a dependency relation between functions, i.e. all underlying functions need to be available in order to provide a certain function. OR-gates on the other hand express redundancy, i.e. the function is available as long as one of its underlying functions is available.

The class model contains default values for availability representing the general belief that a function is available. Naturally, default values for classes on this level of abstraction will have low precision. During the case study, the class model was instantiated to describe a service of the bank: cash withdrawal from ATM machines. The instantiations were performed using the Object Modeler, the dataset already utilized in (Närman et al. 2012) was used to identify objects and their attribute values. To gather this dataset a series of interviews with a middle manager, having the technical responsibility for the bank's ATM system was performed. Figure 6 shows the final model. To the left, the local ATM machine is modeled; the ATM application, the operating system, the PC hardware, and the network connection to the bank's central system are modeled in the center of the figure. This system runs on mainframes, with redundant application servers and databases, localized on different physical sites. Once the model was complete an availability analysis was performed using Metropolis-Hastings sampling, as it was described above. The analysis resulted in an availability prediction of 99.916 % for the *Cash withdrawal with the bank's own card* object representing the investigated service. Figure 7

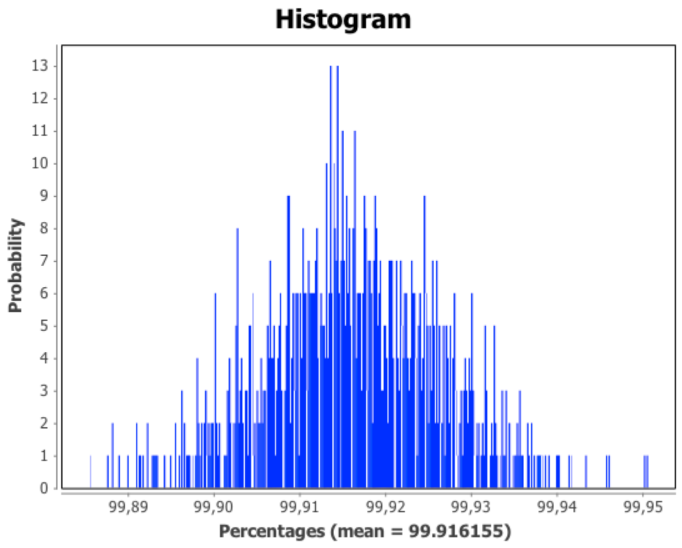


Figure 7 The availability results for the attribute *Cash withdrawal with the bank's own card*

illustrates how this result is visualized in the tool. As this case study was a validation of the assessment framework, the calculated value was compared to the availability found in the logs that the IT department of the bank created periodically. According to the logs the service had an availability of 99.856 %. Once the evaluation was performed, the tool's functionality to trace the impact of an attribute of a certain object was used in order to identify potential for improvement. It was realized that the communication service, used for the communication with the central authorization functionality, had the lowest availability and should be considered for improvements. (Närman et al. 2012) contains more information on the case study and an in-depth discussion of the results.

CONCLUSION AND OUTLOOK

EA is a model-based approach to IT management. A number of EA tools are available. These tools often focus on descriptive aspects of EA and seldom provide advanced analysis capabilities. In contrast to those tools, the presented tool focuses on the analysis of architecture models to support decision-making. The tool supports using the P2AMF and therefore expressing dependencies between properties of created models. Furthermore it considers incompleteness and uncertainty of class and object models. The tool has been used for analyzing a number of different aspects and attributes including business network profitability (P Johnson et al. 2013), interoperability (Ullberg et al. 2010) and modifiability (Österlind et al. 2012).

Future works regarding the presented tool can be separated into two categories. First further development to provide decision support using EA analysis can be considered. Second, it might be investigated whether functionality offered by other tools should be added.

In the first category, potential domains of extensions include result visualization, modeling techniques and automated data collection. It might be investigated how results can be depicted adjusted for different stakeholders. Concerning modeling techniques, one could further investigate how grouping of model parts can be applied in order to make object and class models easier to comprehend. Furthermore, work on the automatic creation of models based on external data sources has been performed (Buschle et al. 2012; Holm et al. 2012). This area might be further investigated too.

The second category of future work, the enhancement of the tool in other areas than decision support and architecture analysis, can be performed considering the tool comparison presented in (Matthes et al. 2008). Based on this, the tool could mainly be improved with respect to three categories: Repository, presentation and administration. Regarding the repository functionality the main area for improvement is the tool's limited capabilities for version handling. From the presentation perspective, there are several more advanced visualization techniques that could be employed. Finally in the administration category, possibilities enabling parallel work could be investigated.

REFERENCES

1. alfabet AG. (2013) planningIT, http://www.alfabet.com/en.
2. BiZZdesign. (2013) BiZZdesign Architect, http://www.bizzdesign.com/tools/bizzdesign-architect/.
3. Buschle, M., Holm, H., Sommestad, T., Ekstedt, M., and Shahzad, K. (2012) A Tool for automatic Enterprise Architecture modeling, *S Olympics: Information Systems in a Diverse World Lecture Notes in Business Information Processing Volume 107*, pp. 1–15.
4. Buschle, M., Ullberg, J., Franke, U., Lagerström, R., and Sommestad, T. (2010) A Tool for Enterprise Architecture Analysis Using the PRM Formalism, *Information Systems Evolution*, 108–121.
5. Department of Defense. (2007) DoD Architecture Framework, version 1.5, .
6. Dunsire, K., O'Neill, T., Denford, M., and Leaney, J. (2005) The ABACUS architectural approach to computer-based system and enterprise evolution, *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*, 62–69.
7. Eastman, C. (1999) Building product models: computer environments, supporting design and construction, .
8. Hastings, W. (1970) Monte Carlo sampling methods using Markov chains and their applications, *Biometrika*.
9. Holm, H., Buschle, M., Lagerström, R., and Ekstedt, M. (2012) Automatic data collection for enterprise architecture models, *Software & Systems Modeling*, 1–17.
10. IBM. (2013) System Architect, http://www-01.ibm.com/software/awdtools/systemarchitect/.
11. Johnson, P., and Ekstedt, M. (2007) *Enterprise architecture: models and analyses for information systems decision making*, Studentlitteratur.
12. Johnson, P., Iacob, M., and Välja, M. (2013) Business Model Risk Analysis: Predicting the Probability of Business Network Profitability, *Enterprise Interoperability Lecture Notes in Business Information Processing* (144), 118–130.
13. Johnson, P., Ullberg, J., Buschle, M., Franke, U., and Khurram, S. (2013) P 2 AMF□: Predictive , Probabilistic Architecture Modeling Framework, *Enterprise Interoperability - Proceedings of the Fifth International IFIP Working Conference, IWEI 2013*.
14. Koller, D. (1999) Probabilistic relational models, *Inductive Logic Programming*.
15. Kurpjuweit, S., and Winter, R. (2007) Viewpoint-based meta model engineering, *Enterprise Modelling and Information Systems Architectures-Concepts and Applications, Proceedings of the 2nd Int'l Workshop EMISA* (Vol. 119), 143–161.
16. Lagerström, R., Franke, U., Johnson, P., and Ullberg, J. (2009) A method for creating enterprise architecture metamodels–applied to systems modifiability analysis, *International Journal of Computer Science and Applications*.

17. Lankhorst, M. (2009) Enterprise architecture at work: Modelling, communication and analysis, .
18. Matthes, F., Buckl, S., Leitel, J., and Schweda, C. (2008) Enterprise Architecture Management Tool Survey 2008, .
19. Närman, P., Buschle, M., and Ekstedt, M. (2013) An enterprise architecture framework for multi-attribute information systems analysis, *Journal of Software and Systems Modeling (online)*.
20. Närman, P., Franke, U., König, J., Buschle, M., and Ekstedt, M. (2012) Enterprise architecture availability analysis using fault trees and stakeholder interviews, .
21. Object Management Group. (2010)*Object Constraint Language, Version 2.2*.
22. Ross, J. W., Weill, P., and Robertson, D. (2006) *Enterprise architecture as strategy: Creating a foundation for business execution*, Harvard Business Press.
23. Software AG. (2013) ARIS IT Architect, http://www.softwareag.com/corporate/products/aris_platform/aris_design/it_architect/overview/default.asp.
24. Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008) *EMF: eclipse modeling framework*, Addison-Wesley Professional.
25. The Open Group. (2008) *TOGAF 2007 edition*, Zaltbommel, Netherlands: Van Haren Publishing.
26. The Standish Group International. (2009) The chaos report, .
27. Troux Technologies. (2013) troux transformation platform, http://www.troux.com/products/troux_platform/.
28. Ullberg, J., Franke, U., Buschle, M., and Johnson, P. (2010) A tool for interoperability analysis of enterprise architecture models using pi-OCL, *Enterprise Interoperability IV*.
29. Walsh, B. (2004) Markov chain monte carlo and gibbs sampling, .
30. Zachman, J. A. (1987) A framework for information systems architecture, *IBM systems journal* (26:3)IBM, 276–292.
31. Österlind, M., Lagerström, R., and Rosell, P. (2012) Assessing Modifiability in Application Services Using Enterprise Architecture Models–A Case Study, *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation Lecture Notes in Business Information Processing Volume 131,*, pp 162–181.