EAAT User Manual

A guide how to use the *Object Modeler*

Dept of Industrial Information and Control Systems, KTH

November 2012

Content

EAAT User manual

This document is a user manual for the Object Modeler part of the Enterprise architecture Assessment (EAAT) tool. For more information about the tool project, confer http://www.ics.kth.se/eaat.

1 Background

The discipline of enterprise architecture advocates the use of models to support decision-making on enterprise-wide information system issues. In order to provide such support, enterprise architecture models should be amenable to analyses of various properties, as e.g. the availability, performance, interoperability, modifiability, and information security of the modeled enterprise information systems. This manual describes a software tool for such analyses. EAAT is an acronym for Enterprise Architecture Analysis Tool and is a software system for modeling and analysis of enterprises and their information systems.

The EAAT tool supports analysis of enterprise architecture models. The tool guides the creaation of enterprise information system scenarios in the form of enterprise architecture models and generates quantitative assessments of the scenarios as they evolve. Assessments can be of various quality attributes, such as information security, interoperability, maintainability, performance, availability, usability, functional suitability, and accuracy.

The EAAT tool consists of two parts, the *Class modeler* for defining the underlying theory for the assessment and the *Object modeler* for modeling enterprises and performing assessments. Theory definition is the activity of identifying which phenomena are relevant for achieving the system properties. For instance, security analysis requires modeling of components such as firewalls, intrusion detection systems, anti malware functions, access right definition and implementation, training of personnel, the existence of business continuity plans and much more. In the Class modeler, the structure and importance of these phenomena is modeled. The Class modeler is thus mainly aimed to be used by researchers. The second part of the tool is the Object modeler, which is used to model instances of system scenarios. The Object modeler uses the theoretical framework developed in the Class modeler. The Object modeler is mainly intended to be used by the industry. While this manual focuses on the *Class modeler*, information about the *Object modeler* can be obtained from the project webpage at http://www.ics.kth.se

1.1 P2AMF: Predictive, Probabilistic Architecture Modeling Framework

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

P2AMF, the Predictive, Probabilistic Architecture Modeling Framework is an extension of OCL for probabilistic assessment and prediction of system qualities. The main feature of P2AMF is its ability to express uncertainties of objects, relations and attributes in the UML-models and perform probabilistic assessments incorporating these uncertainties.

A typical usage of P2AMF would thus be to create a model for predicting, e.g., the availability of a certain type of application. Assume the simple case where the availability of the application is solely dependent on the availability of the redundant servers executing the application; a P2AMF expression might look like this,

context Application :

attribute available : **Boolean** = self.server->exists(s:Server | s.available)

This example demonstrates the similarity between P2AMF and OCL, since the expression is not only a valid P2AMF expression, but also a valid OCL expression. The fist line defines the context of the expression, namely the application. In the second line, the attribute **available** is defined as a function of the availability of the servers that execute it. In the example, it is sufficient that there exists one available server for the application to be available.

In P2AMF, two kinds of uncertainty are introduced. Firstly, attributes may be stochastic. When attributes are instantiated, their values are thus expressed as probability distributions. For instance, the probability distribution of the instance myServer.available might be

P(myServer . available)=0.99

The probability that a myServer instance is avail- able is thus 99%. For a normally distributed attribute operatingCost of the type Real with a mean value of $ 3 500 and a standard deviation of $ 200, the declaration would look like this,

P(myServer.operatingCost)=Normal(3500,200)

i.e. the operating costs of server is normally distributed with mean 3500 and standard deviation 200.

Secondly, the existence of objects and relationships may be uncertain. It may, for instance, be the case that we no longer know whether a specific server is still in service or whether it has been retired. This is a case of object existence uncertainty.

Such uncertainty is specified using an existence at- tribute 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 a 80% chance that the server still exists.

We may also be uncertain of whether myServer is still in the cluster servicing a specific application, i.e. whether there is a connection between the server and the application. Similarly, this relationship uncertainty is specified with an existence attribute E on the relationships.

In this manual the reader will be confronted with P2AMF in three ways: (i) in the form of metamodel attribute specifications, (ii) as metamodel invariants which constrain the way in which the model may be constructed and (iii) as operations which are methods that aid the specification of invariants and attributes.

An example metamodel attribute expression is shown below:

context USAGERELATION

ATTRIBUTE SELF . APPLICATIONWEIGHT : **Real** = GETFUNCTIONALITY ()/ISAFFECTED_1_INV .USE_5_INV . FUNCTIONALITY

This is referring to the class UsageRelation in Figure 2 and specfies that getFunctionality() operation should be utilized. The operation getFunctionality() is specified as follows:

context USAGERELATION

OPERATION GETFUNCTIONALITY () : **Real** = SELF . ISAFFECTED_2 . ASSIGNED_INV−>SELECT (

OCLISKINDOF (APPLICATIONFUNCTION)) . OCLASTYPE (APPLICATIONFUNCTION) . FUNCTIONALITY−>SUM()

where it says that getFunctionality() requires no input, and generates a Real as output according to a statement.

An example invariant, noWriteAndRead, can be found below:

context INTERNALBEHAVIORELEMENT INVARIANT NOWRITEANDREAD = **not** (READ_FUNCTION_INV−>E X I S T S (DO :

PASSIVECOMPONENTSET | WRITE_FUNCTION−> I N C L U D E S (DO)))

This specifies that objects of the class InternalBehaviorElement from Figure 2 are not allowed to both write and read the same data object.

A full exposition of the P2AMF language is beyond the scope Sufficient to say here that the EAAT tool now implements P2AMF using the EMF-OCL plug-in to the Eclipse Modeling Framework and has been employed to implement the metamodels of this paper. The probabilistic aspects are implemented in a Monte Carlo fashion: In every iteration, the stochastic P2AMF 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 P2AMF statement is transformed into a proper OCL statement and evaluated using the EMF-OCL interpreter. The final value returned by the model when queried is a weighted mean of all the iterations.

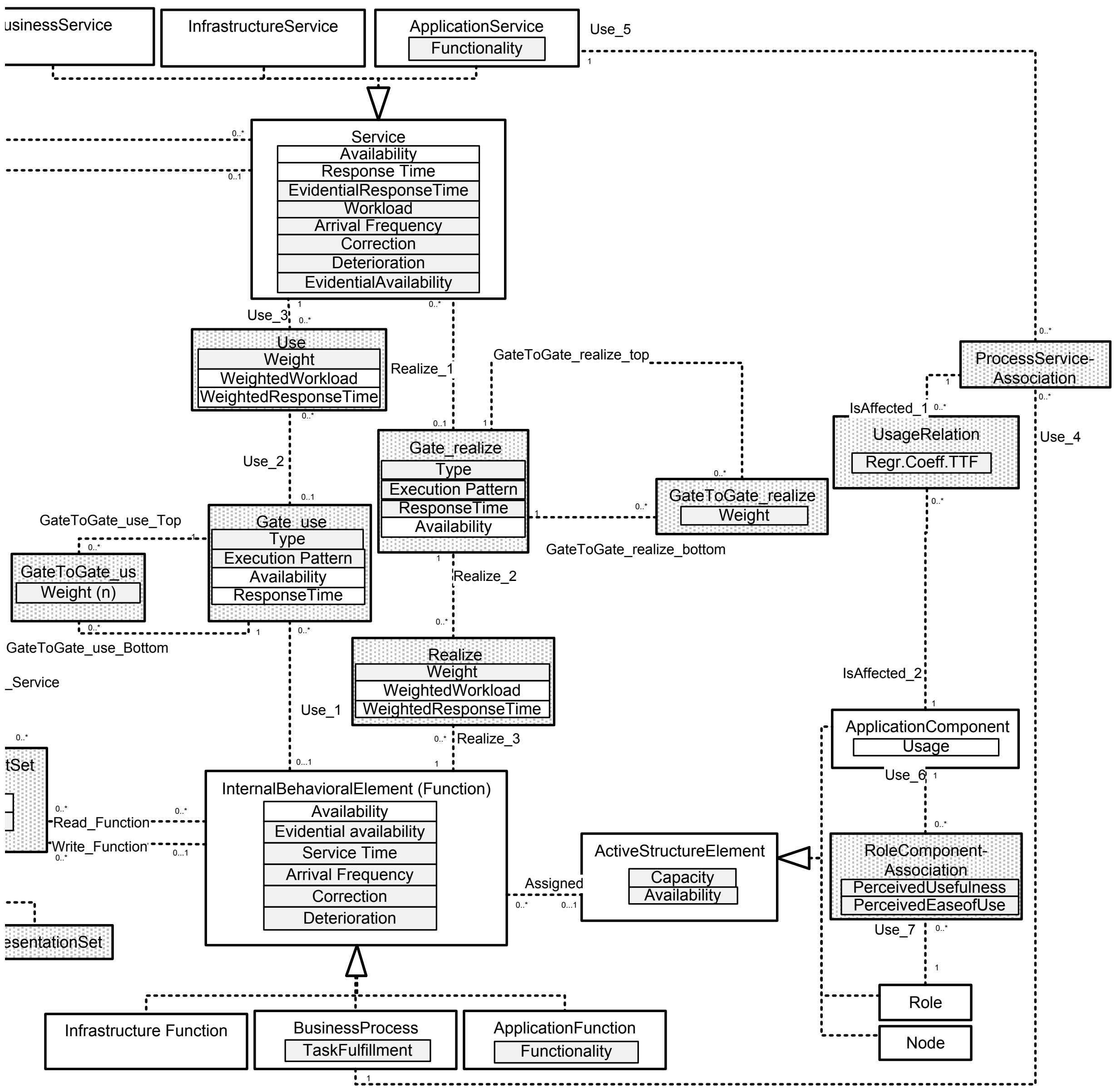


Figure 1 P2AMF Example

In this manual the historic name POCL will be used as a synonym to P2AMF. The wording will be unified with the next major release of the tool and its describing documentation.

1.2 The theory behind the tool

Enterprise architecture models serve several purposes. Kurpjuweit and Winter identify three distinct modeling purposes with regard to information systems, viz. (i) documentation and communication, (ii) analysis and explanation and (iii) design. The present article focuses on the analysis and explanation (which is not to denigrate the usefulness of the others). The reason is that analysis and explanation are closely related to the notion of proper goals for enterprise architecture efforts. For example, a business goal of decreasing downtime costs immediately leads to an analysis interest in availability. This, in turn, defines the modeling needs, e.g. the need to collect data on mean times to failure and repair. In this sense, analysis is at the core of making rational decisions about information systems. An analysis-centric process of enterprise architecture is illustrated in Figure 2. In the first step, assessment scoping, the problem is described in terms of one or a set of potential future scenarios of the enterprise and in terms of the assessment criteria with its theory (the PRM in the figure) to be used for scenario evaluation. In the second step, the scenarios are detailed by a process of evidence collection, resulting in a model (instantiated PRM, in the figure) for each scenario. In the final step, analysis, quantitative values of the models' quality attributes are calculated and the results are then visualized in the form of e.g. enterprise architecture diagrams.

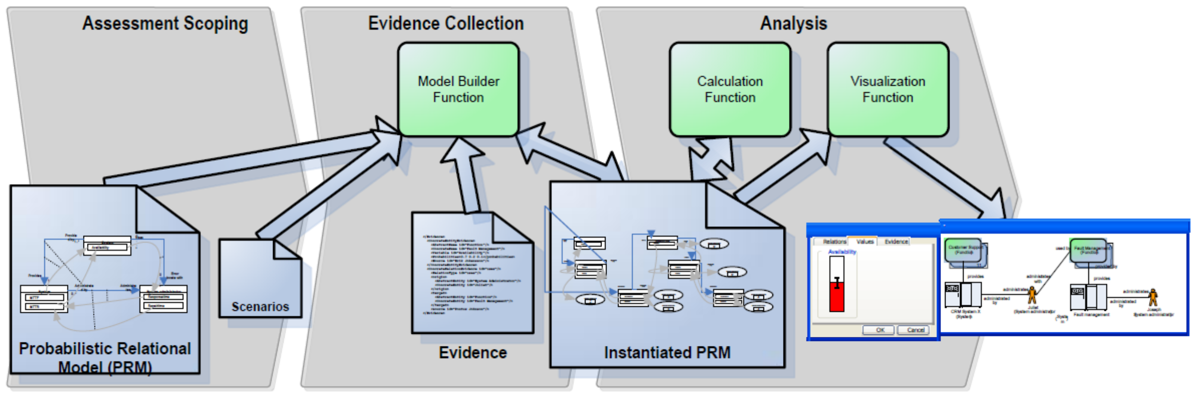


Figure 2 The process of enterprise architecture analysis with three main activities: (i) setting the goal, (ii) collecting evidence and (iii) performing the analysis.

More concretely, assume that a decision maker in an electric utility is contemplating changes related to the configuration of a substation. The modification of a new access control policy would reduce the probability that someone installs malware on a system and thereby reduce the risk that this type of unwanted software is executed. The question for the decision maker is whether this change is feasible or not. As mentioned in the first step assessment scoping the decision maker identifies the available decision alternatives, i.e. the enterprise information system scenarios. In this step, the decision maker also needs to determine how the scenario should be evaluated, i.e. the goal of the assessment. One such goal could be to assess the security of an information system. Other goals could be to assess the availability, interoperability or data quality of the proposed to-be architecture. Often several quality attributes are desirable goals. In this paper, without loss of generality, we simplify the problem to the assessment of security of an electric power-station. Information about the involved systems and their organizational context is required for a good understanding of their data quality. For instance, it is reasonable to believe that a firewall would increase the probability that the system is secure. The availability of the firewall is thus one factor that can affect the security and should therefore be recorded in the scenario model. The decision maker needs to understand what information to gather and also ensure that this information is indeed collected and modeled. Overall, the effort aims to understand which attributes causally influence the selected goal, viz. data quality. It might happen that the attributes identified do not directly influence the goal. If so, an iterative approach can be employed to identify further attributes causally affecting the attributes found in the previous iteration. This iterative process continues until all paths of attributes and causal relations between them, have been broken down into attributes that are directly controllable for the decision maker (cf. Figure 3).

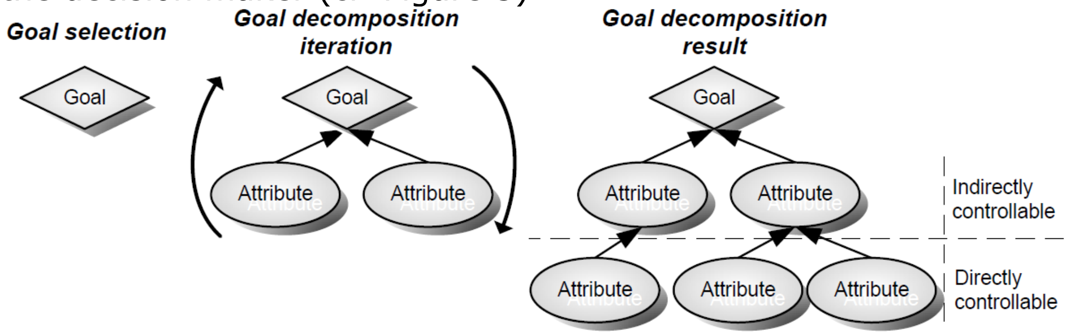


Figure 3 Goal decomposition method.

In the second step collecting evidence the scenarios need to be detailed with actual information to facilitate their analysis of them. Thus, once the appropriate attributes have been set, the corresponding data is collected throughout the organization. In particular, it should be noted here that the collected data will not be perfect. Rather, it risks being incomplete and uncertain. The tool handles this by allowing the user to enter the credibility of the evidence depending on how large the deviations from the true value are judged to be. In the third and final step, performing the analysis, the decision alternatives are analyzed with respect to the goal set e.g. security. The mathematical formalism plays a vital role in this analysis. Using conditional probabilities and Bayes' rule, it is possible to infer the values of the variables in the goal decomposition under different architecture scenarios. By using the PRM formalism, the architecture analysis accounts for two kinds of potential uncertainties: that of the attribute values as well as that of the causal relations as such. Using this analysis framework, the pros and cons of the scenarios can be weighted against each other in order to determine which alternative ought to be preferred.

2 Modeling Object Diagrams in EAAT using the Object Modeler

The theory specified in the Class modeler can be applied in the Object modeler.

2.1 Starting EAAT

EAAT is written in Java to enable platform independence and thus requires Java JRE® to run (if missing, it can be downloaded at http://www.java.com).

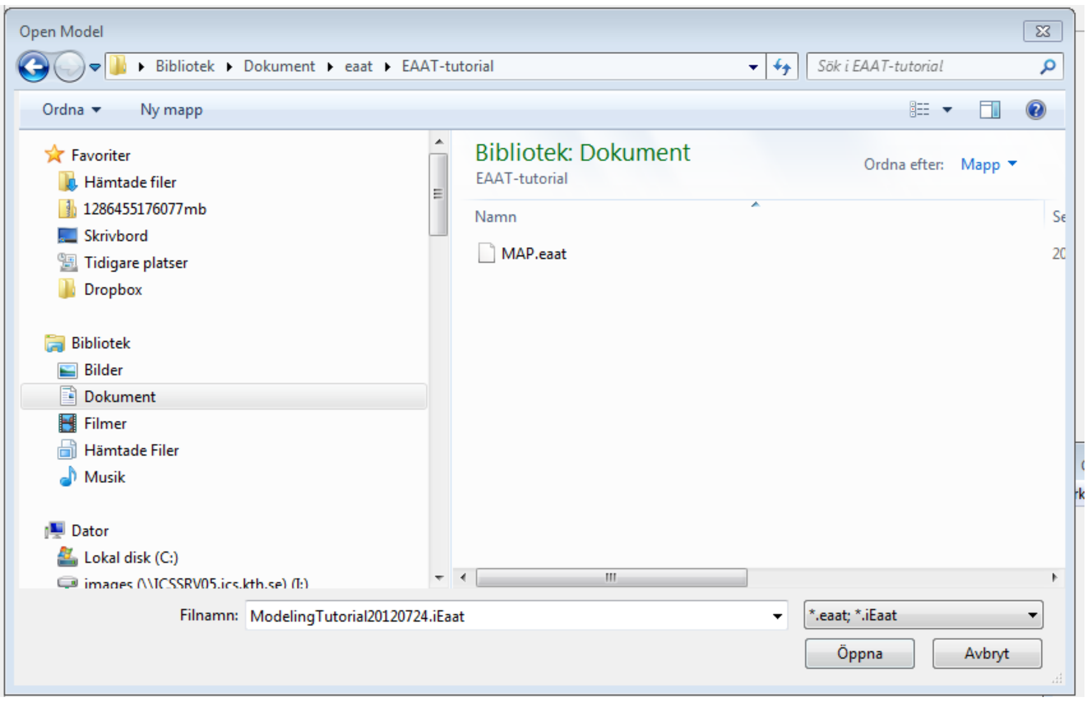
Ensure that you downloaded the latest version of EAAT at http://www.ics.kth.se/eaat

On windows the program is started by executing *ObjectModeler.exe* on

MAC navigate to ObjectModeler.app

2.1.1 Getting familiar with the user interface

When starting the object diagram editor the following dialog is displayed:



MAP.eaat

Filnamn: ModelingTutorial20120724.iEaat

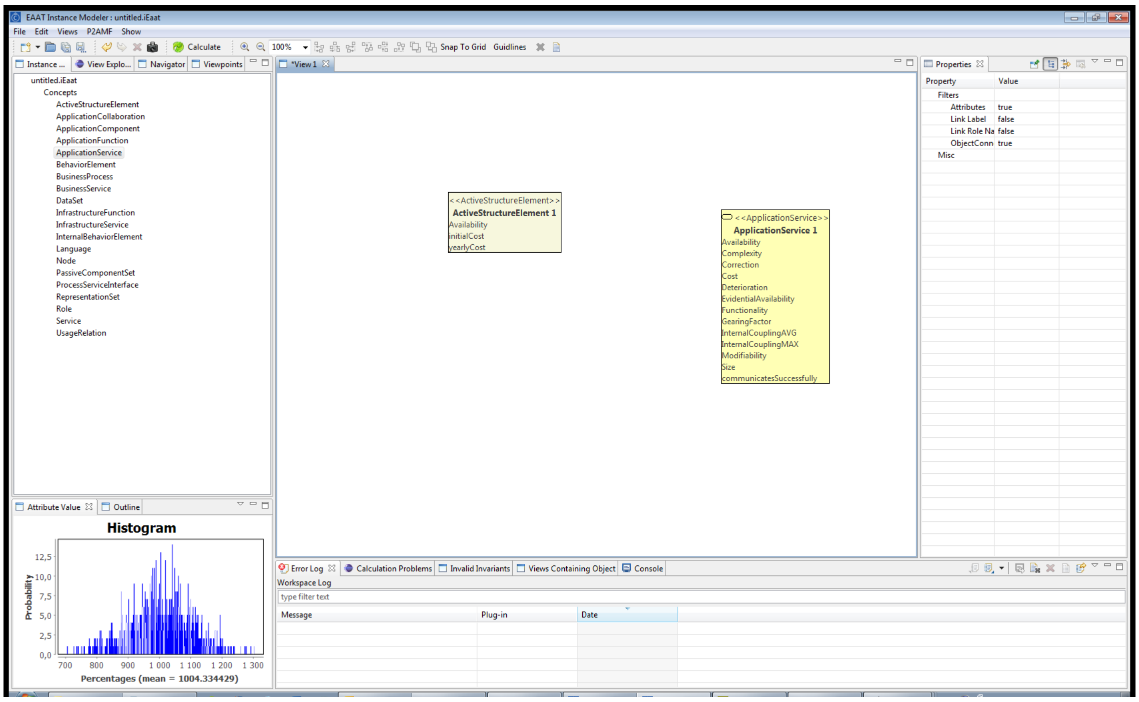
*.eaat; *.iEaat

Figure 4: Start Up Dialog. [1]

In this dialog the user might load a previously created model and continue his work on that. Alternatively one can start with an empty model. To do so a class diagram (*.eaat file) needs to be loaded.

Afterwards the following window is displayed:

[1] The screenshots displayed in this section are based on the Windows version of the tool. The MAC version has a similar user interface, however small differences may occur.



Property	Value
Filters	
Attributes	true
Link Label	false
Link Role Na	false
ObjectConn	true
Misc	

Figure 5: The main window of the EAAT Object modeler with modeling pane, menu bar, tool bar and status bar.

This view is dominated by the large modeling area the white part of the window. Apart from this there is the menu bar and the tool bar at the top of the screen and the info palette on the right. To the right a properties window is presented. In the lower part of the screen several tabs providing different information on the model can be found. Following is a short description of the menu choices and buttons in the tool bar, as well as the info palette.

2.1.2 Menu bar

The menu consists of five top menus: File, Edit, Views, P2AMF and Show. The File menu offers functionality related to the model. We find Open Model, Save, Save … as, Refresh Class model, Export to Excel, Export Class Model and Exit. Refresh Class model allows replacing or updating the used class diagram and export to excel creates a spreadsheet representation of the created object diagram, Export Class Model allows extracting the underlying Class model.

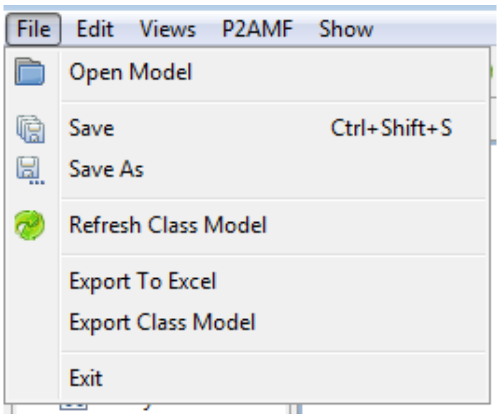


Figure 6 The File menu

The Edit menu allows performing a variety of operations on the model and its objects. These are Undo, Redo, Delete, Cut, Copy, Paste and Select All.

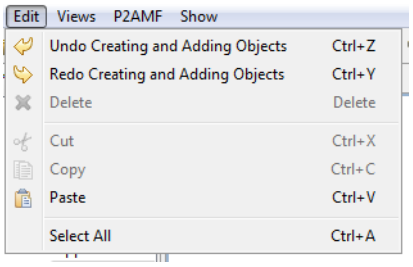


Figure 7 The Edit menu

The Views menu offers functionality for adding new views, as they have been defined within the Class diagram.

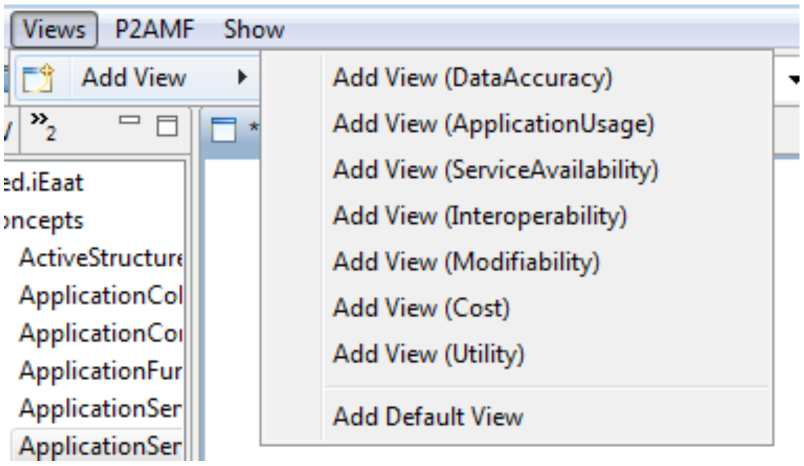


Figure 8 The Views menu

The P2AMF menu collects all operations to the build in calculation mechanism. Options to Calculate the Model, use Deterministic Calculations (without sampling) Check Invariants, Remove all Evidences and perform Configurations are available.

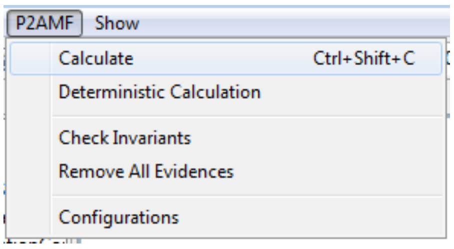


Figure 9 The P2AMF menu

The show menu can be used to display all menus and dialogs that the tool offers. This is particularly important if they have been closed to allocate more space on the screen

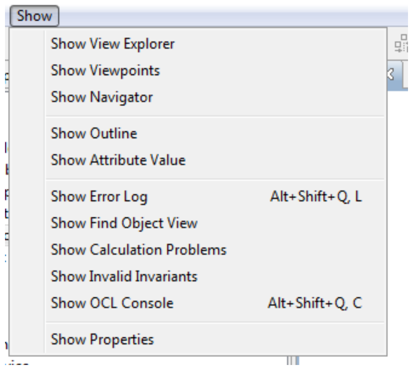


Figure 10 The Show menu

2.1.3 Tool bar

The toolbar, as displayed below, allows for easy access to the most common functions of the tool

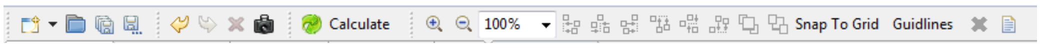


Figure 11: The toolbar of the EAAT Object modeler, containing the most common commands.

Starting from the left there is a for adding new views (instantiating views as they have been defined for the class diagram), opening an existing model, saving the model (including save … as), undoing and redoing a performed operation, deleting a selected object, taking a screenshot of the currently opened model, updating the underlying class diagram, calculating the model, zooming, auto layout, changing the order of displayed instances (bring to front/ send to back) and finally adding notes to the model.

2.2 Info palette

Information on the current model is presented in this information area, located in the lower part of the user interface.

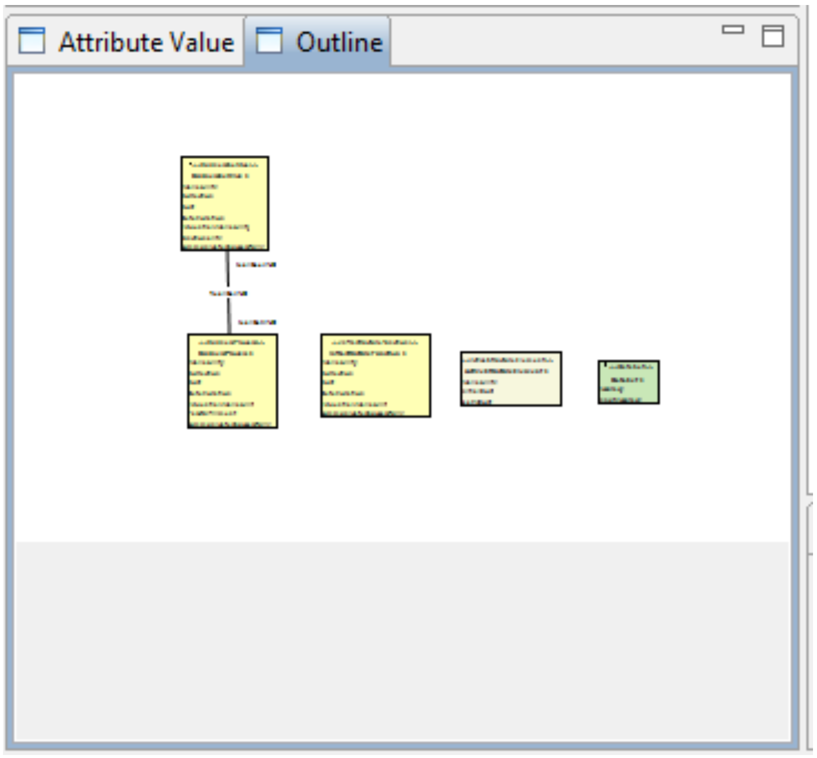


Figure 12: Outline

In one tab an outline providing a global overview can be found. A second tab displays calculation results (c.f. section 2.7).

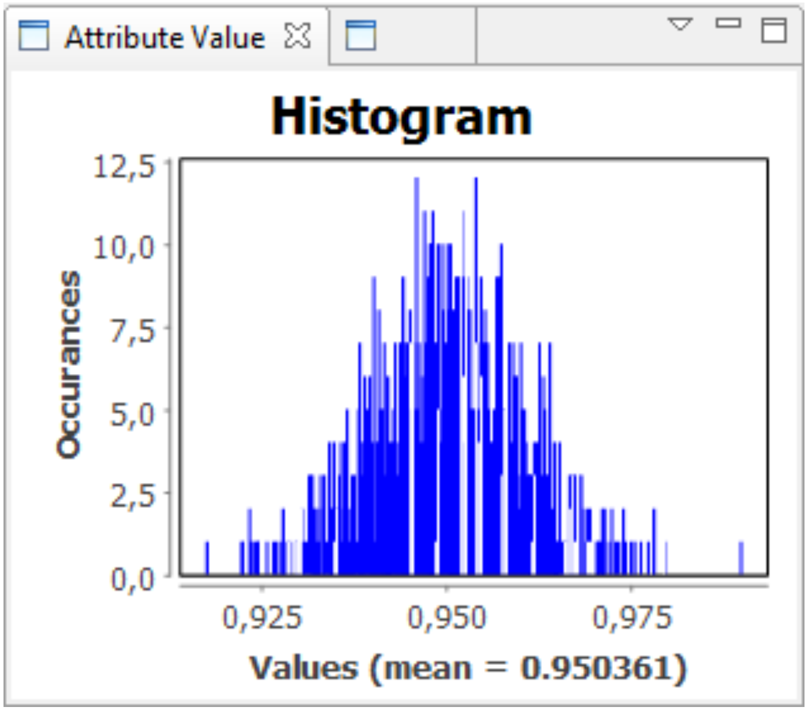


Figure 13 Attribute Value View

Additionally we find a section, divided into tabs, provided extra information on the object diagram.

The following tabs are provided: Error Log, Calculation Problems, Invalid Invariants, Views Containing Object, Console. The first three of them provide additional information in case of unexpected tool behavior during model creation or calculation. The Calculation Problems helps identifying issues with the model (e.g. multiplicities consideration) that need to be resolved in order to being able to calculate the model. The Invalid Invariants shows whether OCL invariants, a feature of P2AMF, are used properly. Yet another view dealing with OCL invariants is the Console, allowing to evaluate and test OCL code and to query the Object Diagram without undergoing the calculation process.

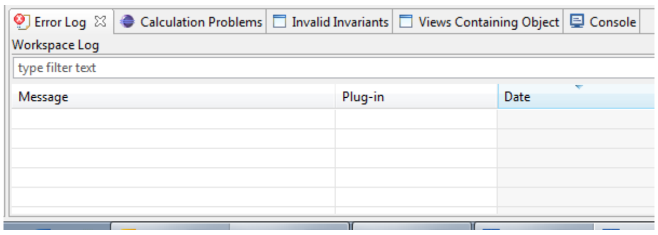
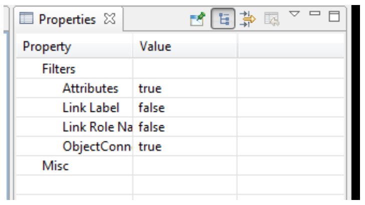


Figure 14 The Info tab section

The most important tab is the Properties tab, located to the right of the user interface of the Object modeler. The Properties tab always displays information of the selected concept whether it is a View, Class, Object or Relationship. These characteristics can be modified directly from the tab.



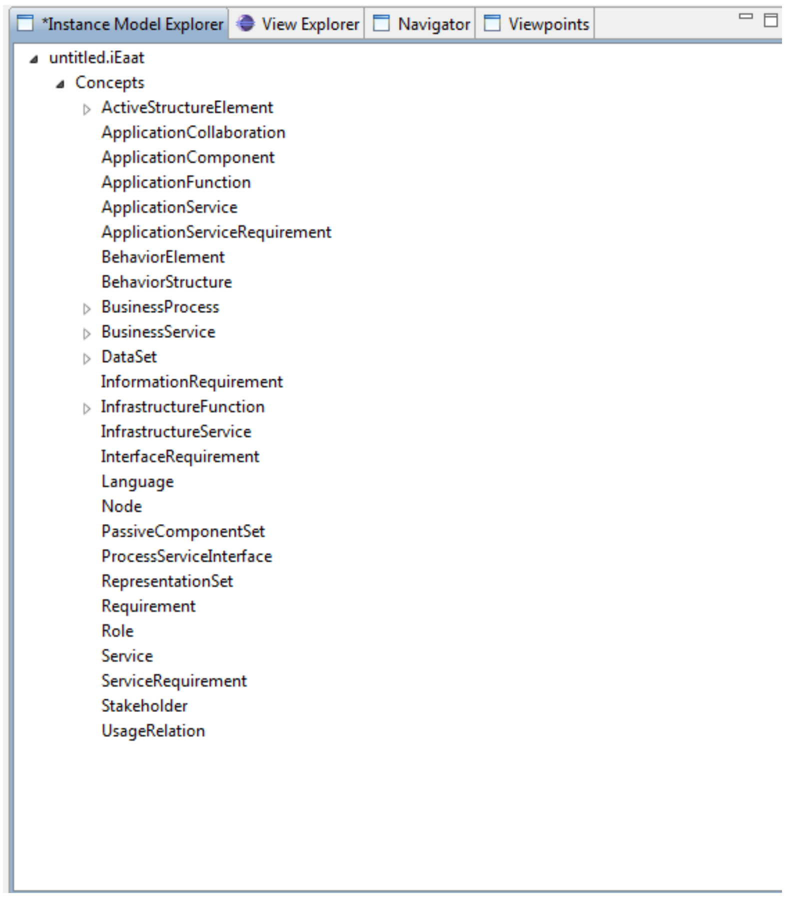
Property	Value
Filters	
Attributes	true
Link Label	false
Link Role Na	false
ObjectConn	true
Misc	

Figure 15 The Properties tab

2.3 Model library

In the upper left corner, information on the created model can be found, as well as the class diagram it is based upon. Four tabs provide an Instance Model Explorer, View Explorer, Navigator and the Viewpoint Explorer.

In the Instance Model Explorer we find an overview over all concepts that are defined in the loaded Class Diagram, as well as the Objects that are instantiating them


*Instance Model Explorer | View Explorer | Navigator | Viewpoints

- untitled.iEaat
  - Concepts
    - ActiveStructureElement
    - ApplicationCollaboration
    - ApplicationComponent
    - ApplicationFunction
    - ApplicationService
    - ApplicationServiceRequirement
    - BehaviorElement
    - BehaviorStructure
    - BusinessProcess
    - BusinessService
    - DataSet
    - InformationRequirement
    - InfrastructureFunction
    - InfrastructureService
    - InterfaceRequirement
    - Language
    - Node
    - PassiveComponentSet
    - ProcessServiceInterface
    - RepresentationSet
    - Requirement
    - Role
    - Service
    - ServiceRequirement
    - Stakeholder
    - UsageRelation

Figure 16 The Instance Model Explorer

In the View Explorer the created views are presented. Each view can be considered as a sub set of model. The User can define as many views as he/she desires.

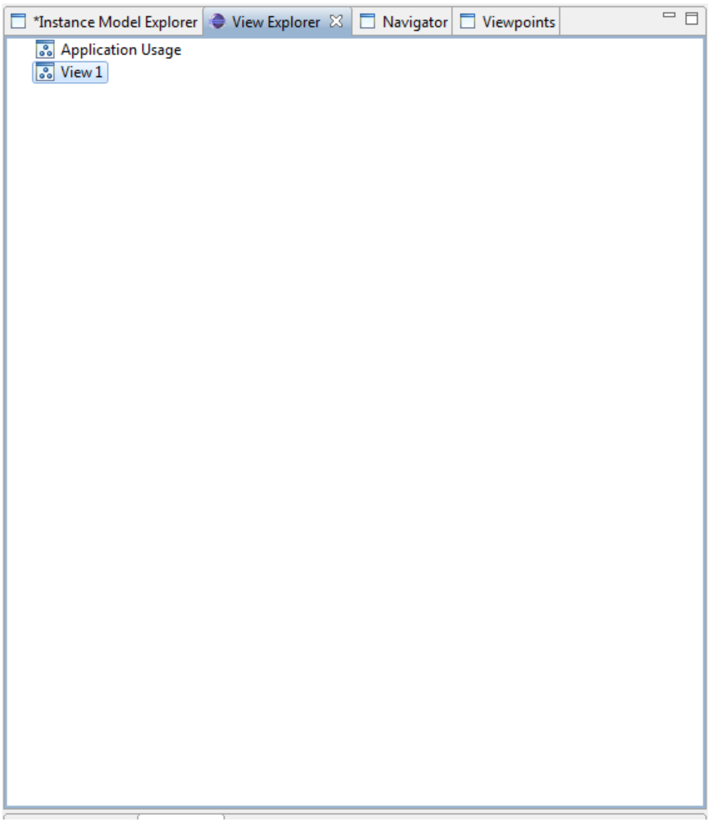
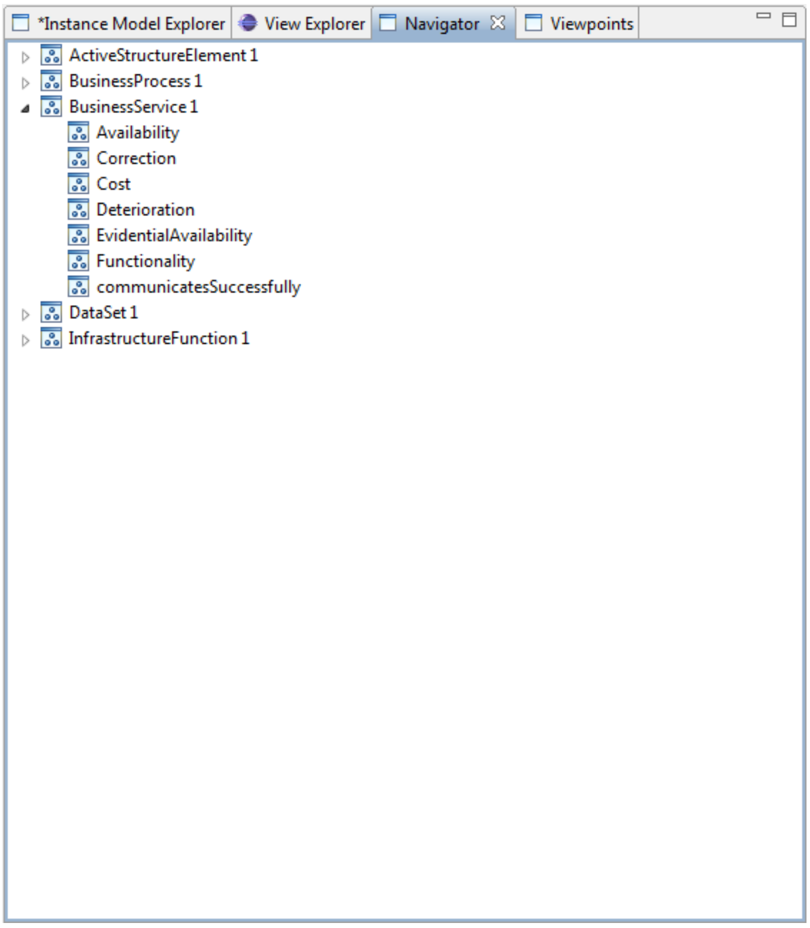


Figure 17 The View Explorer

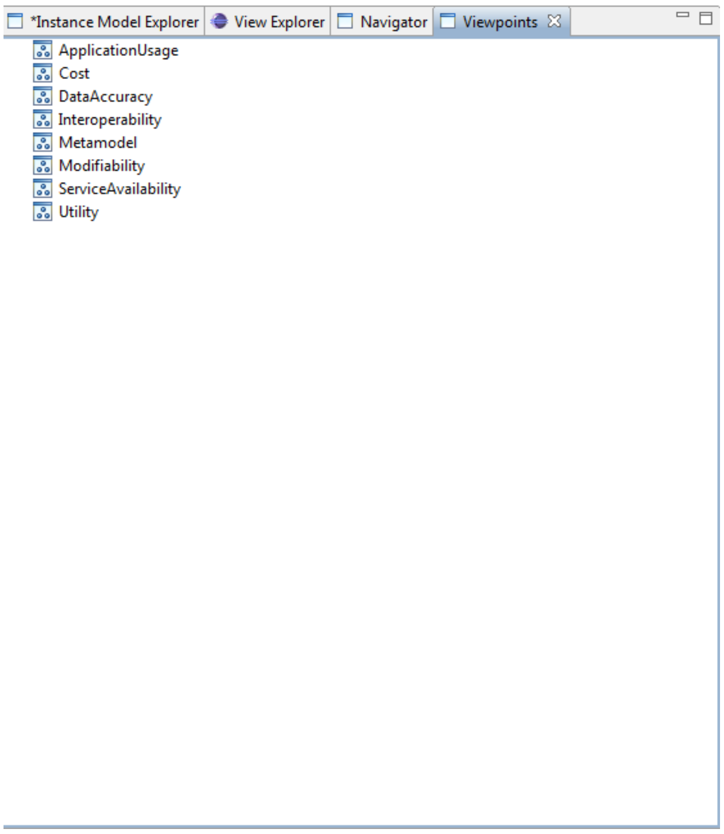
The Navigator provides an overview over all the instances that are part of the currently displayed view.



ActiveStructureElement 1
BusinessProcess 1
BusinessService 1
Availability
Correction
Cost
Deterioration
EvidentialAvailability
Functionality
communicatesSuccessfully
DataSet 1
InfrastructureFunction 1

Figure 18 The Instance Diagram Explorer

Finally the Viewpoint Explorer allows considering the used class diagram and its included viewpoints.



ApplicationUsage
Cost
DataAccuracy
Interoperability
Metamodel
Modifiability
ServiceAvailability
Utility

Figure 19 The Viewpoint Explorer

2.4 Instantiating Classes

Classes are instantiated from the Instance Diagram Explorer. At first the concept to be instantiated needs to be selected followed by a dragging to the currently opened view leads to an addition to the currently displayed model (subset). If an already existing instance is dropped into the modeling area, a reference to it is added to the model subset, whereas new instances are created if a concept that is part of the Class Diagram is dropped into the modeling area.

Instantiating a Class means that the created Object has attributes associated with that type in the class model. For instance, the class “system” might contain attributes such as information security or performance whereas for the class “process”, attributes such as efficiency or cycle time could be appropriate. (Attributes of the classes, as well as all classes are specified by the class model (the theory) – developed in the EAAT Class Modeler.)

2.5 Instantiating relationships

Relationships are created between two objects and which relationships are valid is defined in the underlying class diagram.

To create a relationship the source object needs to be clicked. Second the Alt button needs to be pressed. A drag from source to target, while holding ALT pressed, relates the two objects.

2.6 Adding evidence

Creating instances of classes and relationships builds up the skeleton of the model. This provides basic knowledge for performing the assessment, based upon prior knowledge about the various attributes (i.e. properties) of the model. In order for the assessment to be specific for the case at hand, this skeleton must be augmented with information about the actual states of the attributes in the model.

In most cases, it is very difficult to have direct knowledge about complex attributes such as the information security of an enterprise (otherwise this tool would serve little purpose). However, more low-level attributes such as minimum password length or number of open ports in the firewall can usually be found. While this information is generally not hard to find, it is potentially very resource demanding to actually gather it all. The concept of *evidence* handles this. Evidence is knowledge about the actual states of attributes that is not certain, but has a level of credibility that can be taken into account when performing the analysis. The EAAT Object modeler allows the user to provide evidence(s) regarding the states of every attribute, including the more complex ones, in case such knowledge is available.

Evidence can be added in the Properties tab of a selected attribute (c.f. Figure 14)

2.6.1 Copy & paste of parts of the model

User can copy any object or a collection of objects from any view and the user can paste copied elements to any view as well. This functionality enables fast creation of models. This is done using the short cut CTR+C and CTR+V (CMD on MAC).

2.7 Calculating the model

Once the model is complete in terms of objects, relationships, and evidence on the attributes, it is possible to perform the actual assessment. Since each object model instance has an underlying class model that defines the theory, the user only have to request that the analysis is to take place. The theory is then applied to the model using and all the attributes are evaluated.

The performed calculations can be configured from the properties dialog of the currently created object diagram. At first the object diagram needs to be selected in the Instance Diagram Explorer.

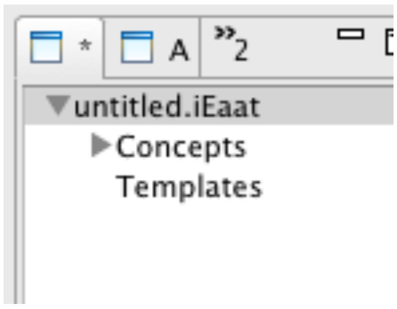
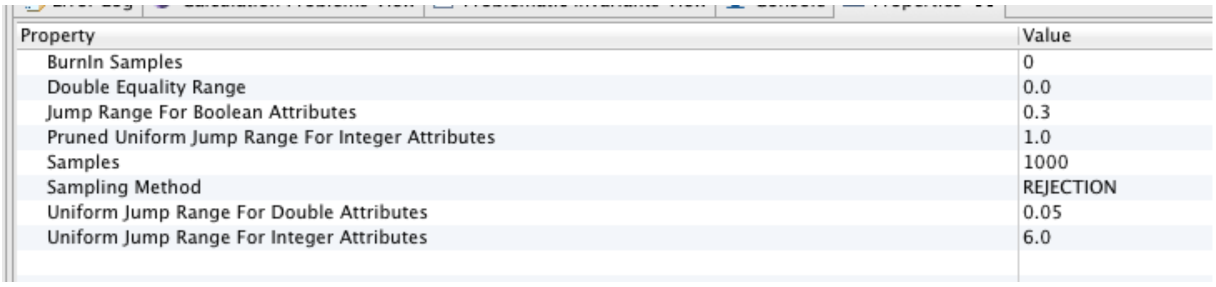


Figure 20 In order to configure the calculations at first the Object Diagram needs to be selected

As a 2nd and final step the calculation can be adopted in the Properties Dialog



Property	Value
BurnIn Samples	0
Double Equality Range	0.0
Jump Range For Boolean Attributes	0.3
Pruned Uniform Jump Range For Integer Attributes	1.0
Samples	1000
Sampling Method	REJECTION
Uniform Jump Range For Double Attributes	0.05
Uniform Jump Range For Integer Attributes	6.0

Figure 21 Calculations are configured from the Object Diagram's Properties Dialog

The Model is calculated by pressing the "calculate" button in the tool bar. To display the result of the assessment consider the Result View, as described in section 2.2 .

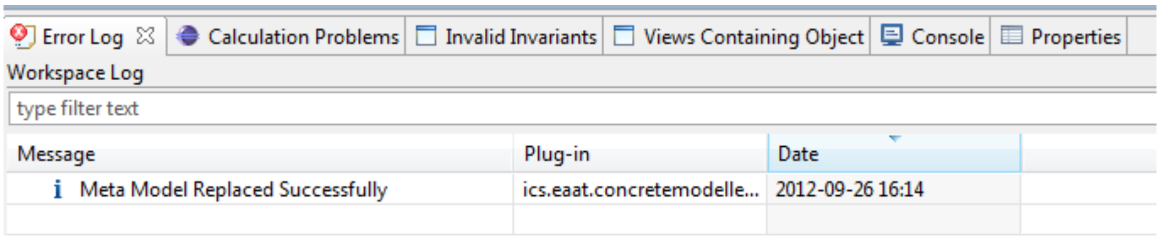
Continuous variables are either presented as single value or as probability distribution, depending on the result. The distributions are presented in terms of:

- Mean value

- Variance
- Skewness
- Kurtosis
- Samples
- Histogram with individual amount of bins

2.8 Replacement of used theory

As mentioned previously, the tool supports the replacement and update of the used theory. The tool tries to match and perform updates as far as possible without user involvement. In case of ambiguities the tool questions the user to solve them.



Message	Plug-in	Date
Meta Model Replaced Successfully	ics.eaat.concretemodelle...	2012-09-26 16:14

Figure 22 If an update of the used theory was performed properly a message is displayed in the Error Log

It is possible to change evidence in case the attributes have been changed.

2.9 Export to excel

The Object modeler of EAAT allows exporting the calculated models to Excel. This allows posting process the results in other tools.

This can be done from the "file"-menu through a usage of the "Export to Excel" option.

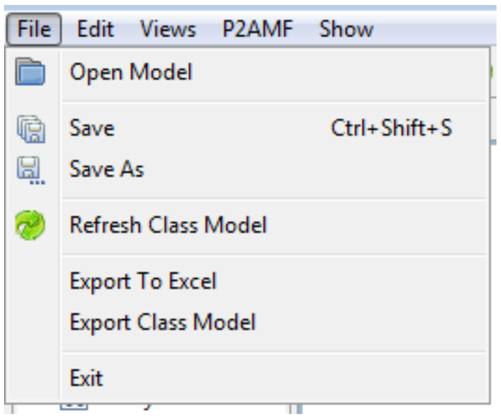


Figure 23: Export to CSV.

In the resulting document each attribute of the model is represented as a row. The complete structure of the document is described in the Appendix.

3 Quick reference

Class modeler. The Class modeler is the part of the EAAT where the underlying theory for enterprise architecture assessments is specified. Here, the concepts and relationships relevant for different kinds of analysis is defined, thus enabling the users of the Object modeler to perform advanced assessments in an automatic, easy fashion.

Class. A class is a category that modeled elements can belong to. When a modeled element in a object model belongs to a class, it means that it has assigned the attributes associated with that class as defined in the class model. For instance, the class "system" might contain attributes such as "information security" or "performance" whereas for the class "process" might have attributes such as "efficiency" or "cycle time".

Credibility. Different data have different credibility, depending on whether the source is reliable, whether it is recently collected etc. When conducting enterprise architecture analyses, it is of greaat importance that models and decisions are not based on flawed or biased data. By requiring the user to specify the credibility of the data used, the EAAT tool manages this aspect of data collection.

Enterprise architecture. The discipline of advocates the use of models to support decision-making on enterprise-wide information system issues. By analyzing the relevant data in a structured, and preferably quantitative, way, better management decisions can be made.

Evidence. Evidence is the data about real world circumstances, collected for the purpose of enterprise architecture analysis. Such evidence is never certain, but rather has a level of credibility that can be taken into account when performing the analysis. The EAAT Object modeler allows the user to provide evidence(s) regarding the states of every attribute, including the more complex ones, should such knowledge be available.

Model. A model is a simplified representation of the real world, specifically designed to capture the aspects relevant for a certain purpose, and leave other aspects out. Enterprise architecture models try to incorporate those feaatures relevant to decision making on enterprise-wide information system issues. The EAAT tool distinguishes two types of models: the class model that speaks of general relationships such as availability and maintenance organizations, and object models that speak of particular companies and situations, such as the availability of system X on company Y. The idea is that the class models are provided by researchers as support for the industry that deals primarily with object models.

Object. An object is a modeling concept, usually referring to something that is part of the real world. Enterprise, CRM system, computer, and project team are all examples of possible objects. Objects have attributes and belong to classes

Object modeler. The Object modeler is used to model instances of system scenarios. The Object modeler uses the theoretical framework developed in the Class modeler to direct and enable complicated enterprise architecture analyses, without a need for the user to be a theoretical expert. The Object modeler is mainly intended to be used by enterprise architects in the industry.

Relationship. Relationships describe how different classes relate to each other. Observable regularities in the real world are modeled as relationships, such as when the "reliability of components decreases, the maintenance costs increase". Relationships based on research are created in the Class modeler, where they serve as templates for analyses in the Object modeler.

View. Represents sub set of model designed to have specialized visualizations. There can be two types of views. First type is normal views which are creaated by user to have some specific visualization and second type views are called analysis view which comes into being to show

4 Appendix

4.1 The structure of the Excel exports

The following schema describes how the resulting excel files are structured:

Name	Meaning
Object-name	The name of the object
Attribute-name	The name of the attribute
Class	The class that the object is an instance of
Attribute-type	The type of the attribute (which right now is either discrete or continuous)
Continuous-value	If the attribute is continuous than the continuous value (after calculation)
State1-name	If the attribute is discrete than the name of the fist state
State1-value	If the attribute is discrete than the probability for that state (after calculation)
…	If the attribute is discrete than the name of the next state
…	If the attribute is discrete than the probability for that state (after calculation)
Staten-name	If the attribute is discrete than the name of the **n**th state
Staten-value	If the attribute is discrete than the probability for that state (after calculation)