ASAAM-T
Aspectual Software Architecture Analysis in Eclipse

F.B. Scholten (0002550)
23-05-2005
Abstract

The ASAAM is a scenario-based architecture evaluation method which analyzes crosscutting concerns in conceptual software architectures. As with most software engineering methods, tool support is needed to properly evaluate and improve the method and its application. Additionally, tool support can provide inspiration for further research in the area of aspectual software architecture design. In the context of a free assignment an Eclipse plugin called ASAAM-T has been developed to support the ASAAM. The plugin has been implemented and designed by the author and the assignment has been supervised by Dr. Bedir Tekinerdogan of the Software Engineering chair of the University of Twente. This report provides an overview of the capabilities of ASAAM-T, its design and implementation, and a roadmap of possible improvements for further development of ASAAM-T.
## Contents

1  **Introduction** ................................................. 4  
   1.1  The Aspectual Software Architecture Analysis Method ........ 4

2  **ASAAAM-T User Guide** ........................................... 5  
   2.1  Architecture Development .................................. 5
   2.2  Scenario Development ...................................... 5
   2.3  Scenario Evaluation ....................................... 5
      2.3.1  Screen Layout ....................................... 5
      2.3.2  Evaluating Scenarios ................................. 8
   2.4  Architectural Component Identification ..................... 8
   2.5  Impact Analysis ........................................... 9

3  **Architectural Design of ASAAAM-T** ............................. 11  
   3.1  Multi-page Editor .......................................... 11
   3.2  Method model ............................................... 12

4  **Implementing ASAAAM-T** ........................................ 13  
   4.1  The Eclipse Modeling Framework ............................ 13
      4.1.1  Code generation with EMF ............................ 13
      4.1.2  EMF code details .................................... 13
      4.1.3  Using the generated code ............................ 14
   4.2  ASAAAM Models Implementation ............................. 15
      4.2.1  Method Model ......................................... 15
      4.2.2  Artifact Model ....................................... 15
      4.2.3  Asaam Evaluation Model ............................... 17
   4.3  Implementing the Multi-page Editor ......................... 17
      4.3.1  Artifact Development ................................ 18
      4.3.2  Method Application .................................. 18
      4.3.3  Result Overview ..................................... 18
   4.4  Improving the current implementation ....................... 18

5  **Conclusions** .................................................. 20  
   5.1  ASAAAM-T Requirements ................................... 20
   5.2  Educational results ....................................... 20

6  **Discussion** .................................................... 21  
   6.1  Usability .................................................. 21
   6.2  ASAAAM Workflow .......................................... 21
1 Introduction

1.1 The Aspectual Software Architecture Analysis Method

A software architecture is a high level design, consisting of components and the relationships between them [1]. During the design of conceptual software architectures, software engineers define the major responsibilities and behaviours of these components. Each component performs one of the major concerns of the application. Adopting the principle of 'separation of concerns' eases maintenance and evolution, since changes corresponding to a single concern will result in changes localized at a single component.

Unfortunately, there are concerns which cannot be localized in a single component. They are called crosscutting concerns. The behaviour corresponding with the concern is scattered over multiple components; the concern crosscuts several components. Tangled and duplicated code in multiple classes can be an indicator of the presence of a cross-cutting concern. Fortunately, crosscutting concerns can be localized using aspects at the programming level, for example by using the Java extension AspectJ [2]. The goal of ASAAM is to identify crosscutting concerns at the architecture design level [3]. The ASAAM-T plugin can be used to perform ASAAM evaluations of conceptual architectures, by defining an architecture and a set of scenarios and following the steps of ASAAM.

This paper is organized in the following manner. The first section presents a user guide for ASAAM-T. It describes how users can define and analyze software architectures with ASAAM-T. The following sections describe the design and implementation of ASAAM-T. A list of knowledge domains is provided which have been used as inspiration for the design of ASAAM. It concludes with suggestions for improving the design of ASAAM-T, as well as introducing new features relevant for further development and provides hints on how these features can be designed. After this overview, the following section describes the implementation aspects of ASAAM-T in the Eclipse environment. It presents several Eclipse plugins and frameworks which were used to implement ASAAM-T. Additionally it shows how the existing implementation can be improved. Finally, we will provide an overview of results of the ASAAM-T assignment.
2 ASAAM-T User Guide

This section provides a short tutorial for performing ASAAM evaluations with ASAAM-T. To start an ASAAM-T evaluation, open the navigator window and create a file with an ‘.eval’ extension in a project. If necessary create a new project first. Next, open the file in the default ASAAM-T editor. ASAAM-T is a multi-page editor, which has a separate page for each major activity in ASAAM. The following sections provide a walkthrough of each of these activities.

2.1 Architecture Development

The first activity of ASAAM is architecture development. Figure 1 shows the page for developing a candidate architecture. This page provides an interface to create, remove and modify architectural components. By clicking the ‘add component’ button, a new component is added the viewer on the left of the screen. The ‘remove’ and ‘clear’ buttons can be used to remove all or a few selected components. The viewer in the section ‘architectural components’ consists of two columns. The first column shows an id, the other column shows the components name. The user can change the components name by clicking on the text in the second column. When a component is selected, the section ‘architectural component details’ the right part of the screen becomes visible. This section provides an interface to edit the components description. Naturally, the architectural components defined through this interface originate from a previous architecture development phase.

2.2 Scenario Development

The second type of input artifact of ASAAM are scenarios. The user interface resembles the user interface on the previous page. You can create scenarios, similar to the way architectural components are created in the architecture development page. Scenarios consist of a short description, as well as a more detailed description. The detailed description provides additional contextual information about the scenario interaction. Only the short description is used in the latter parts of evaluation.

2.3 Scenario Evaluation

2.3.1 Screen Layout

The scenario evaluation page provides an interface for evaluation scenarios, according to the ASAAM rules R1 through R6. The top of the screen shows a viewer containing all scenarios defined through the previous page. The columns show the scenarios id, its description, its ASAAM type, the number of interacting components and its crosscutting nature. By selecting a scenario from this viewer, the lower half of the screen becomes active. The middle of the page shows the second viewer containing all components defined through the architecture development page. The columns in this viewer show the components id,
Figure 1: The user defines a candidate architecture
Figure 2: The user creates several scenarios
Figure 3: The user evaluates individual scenarios

its name, a dropdown box containing interaction types and a textual impact. This viewer is used for the scenario evaluation activity. The bottom part of the page shows the question ‘Is the scenario scattered?’ with ‘yes’ and ‘no’ buttons.

### 2.3.2 Evaluating Scenarios

To evaluate a scenario, the user first selects a scenario from the first viewer. Then the user turns to the second viewer and by selecting ‘direct’ or ‘indirect’ from the dropdown box in the second column, the user can decide the what kind of interaction a scenario has with the component on that row. If the empty field in the dropdown box is selected, the scenario has no interaction with the corresponding component. After this activity, the user answers ‘yes’ or ‘no’ to question which asks if the scenario is scattered. After all scenarios have been evaluated, each scenario has been identified with an ASAAM-T scenario type.

### 2.4 Architectural Component Identification

This page shows the interacting scenarios for each component. The viewer at the top of the page has four columns, which show the components id, its name, its ASAAM type and the number of interacting scenarios. By selecting a
component from the viewer, three additional viewer appear on the middle of the screen. These viewers show direct, indirect and aspectual scenarios, interacting at the selected component. Based on this information the viewer can answer 'yes' or 'no' to several questions, which ask whether the component performs semantically close scenarios, and whether the component can be decomposed. This entire activity activates ASAAM rules R7 through R16.

2.5 Impact Analysis

The final page shows the overall impact analysis of the architecture. The columns of the viewer represent components and the rows are scenario ids. An 'D' in the viewer means that a scenario interacts directly at a component, while a 'T' means an indirect interaction. If a red ball is present, it means the interacting scenario aspectual.
Figure 5: The impact analysis shows the impact of scenarios on components
3 Architectural Design of ASAAM-T

ASAAM-T is a CASE tool for evaluating software architectures. A CASE tool provides an interface to apply a certain software development method. The two main concerns for ASAAM-T are method application and user interaction. To support these concerns, ASAAM-T is built as a layered system with two layers, as depicted in Figure 6. The upper layer is a multi-page editor, the lower layer is the method model, responsible for method execution and updating the multi-page editor. The method model is the most important part of ASAAM-T, as it directly represents ASAAM. We will discuss the design of the method model and the multi-page editor in the next sections.

3.1 Multi-page Editor

The multi-page editor provides an interface for user interaction in ASAAM-T. Each page in the editor provides an interface for a subactivity in ASAAM, such as candidate architecture development, scenario evaluation, and so on. All pages provide a different view of the method model. The user can switch pages to continue one of the different activities. All user interface components are refreshed if the method model changes state. A detailed overview of the components contained in the multi-page editor can be found in the section 4.3.
3.2 Method model

The domain of method engineering has been used as a source of inspiration for ASAAM and its concepts have been used to design ASAAM-T. Method engineering is the study of designing methods. Several concepts from [4] have been used as a basis for our method model. It demonstrates the concepts ‘method’, ‘artifact’, ‘rule’ and ‘process’. The goal of methods is to transform or manipulate artifacts using method rules. Artifacts are the concepts relevant to the domain of interest. In the case of ASAAM, the domain is aspectual architecture evaluation and the artifacts are multiple types of scenarios and components. Method rules manipulate artifacts through actions based on conditions of artifact properties. These method rules are evaluated in a certain order. A process determines the order of rule evaluation. Figure 7 depicts the method model used in the design of ASAAM-T.
4 Implementing ASAAM-T

This section describes implementation aspects of ASAAM-T. Since ASAAM-T uses a lot of code generated by the Eclipse Modeling Framework plugin, we will first provide an introduction to this framework. After this introduction, we will describe how the different components of ASAAM-T are implemented. That part is divided in three sections. The first section describes the implementation of the method model, which implements the core of ASAAM. The second section describes the artifact model in more detail. Finally, the implementation of the overall ASAAM evaluation is described. At the end we will give an overview of possible improvements of the current implementation.

4.1 The Eclipse Modeling Framework

4.1.1 Code generation with EMF

EMF is capable of generating code from annotated Java interfaces or from an Ecore model created with the Ecore editor, available through the EMF plugin [5]. An Ecore model is an XMI document which specifies classes, methods, attributes and relationships between classes, such as inheritance and composition. In the development of ASAAM-T we have used the Ecore editor from EMF to create the models described in later sections. After the model definition, a generator model, or genmodel, is used to generate Java code to be used as a base model for the ASAAM-T plugin. The Java interfaces generated from the Ecore model can be manually changed by inserting new methods or member variables and inserting a comment /* @generated NOT */. In the next section we will describe the nature of the code that is generated by the EMF genmodel. After this, we provide short examples on how we have used the generated code in ASAAM-T.

4.1.2 EMF code details

The code generation process of EMF can be configured through the genmodel file. Several features can be toggled, such as insertion of notification code, choice of types for reference values, and so on. EMF can generate 3 plugins containing generated code, the model plugin, the edit plugin and the editor plugin. The model plugin contains a direct implementation of the Java interfaces, which respects cardinalities of reference values. For each attribute, get and set methods are generated. Notification code is inserted around every change of attribute or reference values. This code involves calling the eNotify method of the org.eclipse.emf.common.notify interface with certain parameters. In case of multiplicity-many references, get-methods returns an org.eclipse.emf.common.util.EList, containing the referenced objects. The model plugin also contains a Factory class which has factory methods for each class in the model. The edit plugin, com.asaam.architectureevaluation.emfmodel.edit, contains an implementation of adapter classes, used for editing and viewing the generated model objects. A more thorough description of workings of the EMF edit framework can be found.
in [6]. The most important classes are the ItemProvider classes. ItemProviders can wrap model objects in a way that they can be used in combination with JFace viewers, such as TreeViewers or TableViewers. ItemProviders have several responsibilities, as shown in [6]. Most importantly, they delegate notification of changes in a model object to the JFace viewer, so that the viewer updates if a model element changes state. Second, they implement one of the many ContentProvider and LabelProvider interfaces, which define the textlabels and icons that are shown in the JFace viewers. The third plugin contains code for a complete editor. This editor can be used to create & edit model elements. Because the ASAAM-T editor was going to very different than the generated editor we have not used the code from this plugin.

4.1.3 Using the generated code

ASAAM-T uses a lot of JFace viewers for displaying collections of artifacts, such as a selection of scenarios. Figure 8 shows an example on how to initialize a JFace viewer with generated EMF model objects. A JFace viewer can be initialized by setting ContentProviders and LabelProviders and setting a container object, in this case the architecture, as the input. Normally we would have to create the provider classes ourselves, but we can use the AdapterFactoryContentProvider and AdapterFactoryLabelProvider classes to wrap our generated ItemProvider-AdapterFactory class, the ArchitectureEvaluationItemProviderAdapterFactory. The ItemProviderAdapterFactory has references to all ItemProviders. If the content of the viewer needs to be refreshed, the AdapterFactoryContentProvider delegates this command to the wrapped ItemProviderAdapterFactory which delegates the command in turn to the ItemProvider object of the model object, which computes a new text label using the model object. This mechanism works in the same way as with AdapterFactoryLabelProviders to generate new icons.

Usually, an ItemProviderAdapterFactory needs to delegate to other ItemProviders to create icons and text labels. In case of ASAAM-T we have many viewers and pages which each need different icons and text labels. For example, the viewers in the scenario evaluation page have different column names and values than the viewers in the component identification page, even though the input to the viewers is the same, the architecture object. The problem is that we need multiple presentations of a model, while the AdapterFactory design used in EMF only allows a single presentation for a type. We have added a setType and getType method to parameterize the workings of the factory ArchitectureEvaluationItemProviderAdapterFactory. This is not an elegant solution, as it goes against the design considerations of the AdapterFactory. The author has not found a good solution for this problem, using the current documentation and knowledge about EMF. However, the EMF framework is being used more and more and many solutions will present themselves as more different types of modelling problems will be tackled.
Figure 8: JFace Viewer initialization

```java
ArchitectureEvaluationItemProviderAdapterFactory af = new ArchitectureEvaluationItemProviderAdapterFactory();
af.setType(ArchitectureEvaluationItemProviderAdapterFactory.COMPONENT_ANALYSIS);
archViewer.setContentProvider(new AdapterFactoryContentProvider(af));
archViewer.setLabelProvider(new AdapterFactoryLabelProvider(af));
archViewer.setUseHashlookup(true);
archViewer.setInput(eval.getArchitecture());
```

Figure 9: Asaam Method Interfaces

![Diagram of Asaam Method Interfaces]

4.2 ASAAM Models Implementation

4.2.1 Method Model

The method model described in 3.2 was implemented using the interfaces described in Figure 9. We could define the interfaces of the method model almost directly in the EMF ecore editor. A few classes needed to be altered manually after code generation. For example, method rules needed to be instantiated in the AsaamMethod class. Rules needed be registered to a method, by declaring which artifact it will manipulate when a condition is met.

4.2.2 Artifact Model

Figure 10 describes some more detailed interfaces of the artifacts. The artifact model consists of the superinterface AsaamArtifact, and two subinterfaces AsaamComponent and AsaamScenario. These interfaces define the properties
necessary for an ASAAM evaluation, such as the ASAAM types & properties and descriptions.

Finally, the fourth interface in this model is the MappedScenario subinterface which inherits from AsaamScenario. This interface defines impact methods for AsaamScenario artifacts which have been mapped to components. Since scenarios can be evaluated to multiple components, impact information needs to be kept for each specific evaluation. We have solved this problem by creating a decorator which wraps the original scenario and introduces localized state, the impact of the scenario.
4.2.3 Asaam Evaluation Model

In the previous sections we have described the detailed definition of artifacts and ASAAM method interfaces. An actual ASAAM evaluation involves several scenarios and architectural components. This section describes interfaces for working with complete ASAAM evaluations. These are top level interfaces which act as containers for several artifacts in ASAAM. The graphical editor uses these interfaces as an access point to the entire ASAAM evaluation. Figure 11 shows the interfaces for Architecture, ScenarioSelection and ArchitectureEvaluation.

The AsaamEvaluation interface is the top level interface for an ASAAM evaluation. It provides methods which access the AsaamArchitecture and AsaamScenarioSelection interfaces. Additionally, it defined methods which compute metrics about the evaluation. The Architecture interface provides methods for adding, removing and retrieving architectural components. The ScenarioSelection interface provides similar functionality for AsaamScenarios.

4.3 Implementing the Multi-page Editor

User interaction is the most important part of applying any software engineering method. Several requirements were needed for the ASAAM-T user interface. First, the user must have an interface for creating artifacts. Second, the user needs an interface to inspect and manipulate artifact properties to apply the ASAAM. The process of applying an ASAAM evaluation consists of artifact development, method application and presenting a results overview. We have used a multi-page editor for implementing the ASAAM-T GUI. These pages use SWT buttons and textfields and JFace viewers. SWT is the standard GUI toolkit on which Eclipse itself is built. JFace is a high level API build on SWT,
used to easy GUI programming in Eclipse. Additionally we used the Form API from Eclipse 3.0 to create a flat layout, which resembles the Eclipse plug-in editor [7]. For each of the mentioned process steps we will describe the user interface in more detailed fashion.

4.3.1 Artifact Development

ASAAM uses two types of input artifacts: scenario and architectural component. In the artifact development phase the user needs to create several artifacts for both types, so we have decided to create a separate page for each type. Each page contains a Master-Details structure. The master part contains a viewer of the created artifacts with buttons to add and remove artifacts. The details part shows the properties of the selected artifact in the viewer. The user can manipulate properties in the details part.

4.3.2 Method Application

The method application phase of ASAAM-T consists of the third and fourth page. The ASAAM method rules manipulate multiple scenario and component artifacts at the same time and create relationships between artifacts. For example, the user needs to view all direct scenarios of a component, as well as all components interacting at a scenario. As a consequence, the scenario and component artifacts each get their own page. These pages consist of several viewers for scenario and component types. JFace filters are used to view only certain ASAAM artifact types.

4.3.3 Result Overview

The method application pages contain a partial overview of the results of the ASAAM method. However, one final page is needed to show the complete overview of the relationships between scenarios and components. This page consists of a grid which shows interactions between components and scenarios.

4.4 Improving the current implementation

The current implementation of the ASAAM-T GUI consists of a combination of SWT and JFace controls to represent a view of the underlying method model. The JFace components use a clearly defined protocol for communicating with the method model. The object in the method model objects notifies their item-providers, which in turn notify their JFace viewers. The SWT controls however do not use such a clearly defined protocol for communication with the method model. As a consequence, notification and manipulation of the model through SWT controls had to be implemented in an ad hoc manner. An improvement to the existing implementation would be to create a stable model view controller design. This will require a view which is uniformly updated when the model send notifications.
5 Conclusions

5.1 ASAAM-T Requirements

The delivered tool supports only a small subset of the requirements originally presented in the requirements specification. Supporting tools, such as a graphical architecture modeler, as well as an architecture refactoring tool were cancelled due to time and scope constrains. The tool can be used to evaluate simple software architectures according to the ASAAM method. All the associated method rules are implemented accordingly, as described in [3]. Software architectures and scenario selections created with the corresponding generated EMF editors can be saved and loaded into the evaluation tool. Graphical Architecture modeling seemed to be beyond the scope of the assignment due to time constraints. Furthermore, the use of composite architectural components is not implemented.

5.2 Educational results

During this assignment the author has acquired a considerable amount of experience with developing in the Eclipse platform. Eclipse provides many frameworks to develop sophisticated GUIs. Some components them have been used extensively, such as JFace, SWT & EMF, while many others are not used at all, such as Wizards and Views. We noticed that the developing in the Eclipse environment had a greater learning curve than expected. Besides learning to program in the Eclipse environment, the author has learned to use the JUnit testing framework. During this assignment much experience has gained on not only implementing applications in a new environment, the Eclipse environment, but also with dealing with the complexity of a problem domain. The main steps of the process of developing the tool are simultaneous investigation of possibilities and limitations of the eclipse platform on the one hand and finding solutions to problems from the problem domain of the tool on the other hand. One important lesson the author learned is that with every time one encounters a new technology, time is needed to practice and learn to use it. In this project the time to learn to use and implement tools in Eclipse was underestimated.
6 Discussion

This last section proposes several suggestions for additional features of and ideas about the ASAAM-T. During the development of ASAAM-T, numerous additional requirements were identified that would be relevant for future development. The goal of this section is to provide ways in which ASAAM-T can evolve to a more complete CASE tool. There are many possible directions in which ASAAM-T can evolve. We now present several features and concepts from different knowledge domains which can be implemented in the future. The collection of items mentioned is by no means exhaustive and is meant as a general direction for further development of ASAAM-T.

6.1 Usability

Methods outline a process to achieve a certain result. In case of ASAAM, the result is an architecture evaluation with respect to crosscutting concerns or aspects. The user of ASAAM-T continually needs to view and manipulate information. When architectures become complex and the scenario selection large, the issue of searching information becomes relevant. If a software engineer performs an architecture analysis he will need to view information on different levels of granularity. In case of scenario evaluation he needs to analyze a scenario one at the time. To assess impact on several components, multiple descriptions have to be viewed and an impact must be added. At each point the software engineer should be aware which choices he has made and yet has to make. One possibility is that the software engineer may want to query the scenario database or architecture to for example quickly find architectural components which contain descriptions mentioning keywords such as 'networking' or 'graphical user interfaces'. To support these requirements additional research may be needed to use create a scalable user interface for ASAAM-T.

6.2 ASAAM Workflow

ASAAM-T is built on concepts from the method domain model, described in the introductory paper of ASAAM [ASAAM]. This domain model uses concepts rule, method, process and artifact. This domain model provides a way to structure rule evaluation in ASAAM-T. While ASAAM is a method which exists as a separate conceptual framework, ASAAM-T is a CASE tool which requires additional features to support user activity. While many features are implemented in the current ASAAM-T version, many more features are necessary to create sophisticated user support. Examples of such features are contextual menus [8]. The former patterns are related to human computer interaction or user interface design domains.

Another concept which may be relevant for user involvement in ASAAM-T is a workflow. Workflows integrate user activity with well defined methods. Workflows use state machines to formally describe the underlying method and
uses validators and action providers to allow feedback and control to users to manage the workflow. The Workflow perl module provides all these features and can be used as an example of how to model workflows [9]. The workflow model can be thought of as an extension to the method model described in [3]. It explicitly models constructs which allow the user to control methods, while the method model only defines rule evaluation and method structure, not how rules are activated and how method feedback can be shown to the user or the component controlling the workflow.
References


