ALAT

a new authoring environment for GALE

Boereboom, W.

Award date:
2016

Eindhoven University of Technology

Department of Mathematics and Computer Science

ALAT

A new authoring environment for GALE

Wouter Boereboom

Supervisor:

Prof. dr. P.M.E. De Bra

Tutor:

Dr. ir. Natalia Stash

Graduation Committee:

Prof. dr. P.M.E. De Bra

Dr. ir. Natalia Stash

dr. C. Huizing

Eindhoven, January 2016

Preface

This master thesis is the result of my graduation project performed internally at the Eindhoven University of Technology, at the web engineering group. This project completes my Computer Science & Engineering study in Eindhoven.

First off I would like to thank Prof. Dr. P.M.E. De Bra for his support, advice and numerous meetings and proofreads throughout my entire graduation project.
Numerous meetings with members of the university and “De Roode Kikker” have been a great inspiration and resource in the creation of ALAT. I would like to thank Dr. ir. Natalia Stash for her advice and guidance throughout these meetings.

The staff off “De Roode kikker” has also been a great source of new ideas and provided a new perspective on authoring adaptive applications. They have been worthy sparring partners throughout this project.

During these meetings the interface designs used for ALAT have been created based on discussions about what an authoring interface should look and feel like. A big thanks to

Yuexu Chen for creating these wonderful designs and for putting up with our off-track discussions and Dutch banter during all these meetings.

Last but not least, a big thanks to all my friends and family who have supported me throughout this graduation process. Thank you for your motivation, advice, proofreading and discussions.

Wouter Boereboom,
January, 2016

Abstract

GALE [SmBr11] (GRAPPLE Adaptive Learning Engine) is a “truly generic and general purpose adaptive hypermedia engine” with which a wide variety of adaptation techniques can be implemented. It is used for teaching about adaptive hypermedia and in various projects conducted at the Eindhoven University of Technology.

A major challenge in the creation of adaptive hypermedia applications, often called authoring, lies in its complexity. Authoring adaptive applications is often tricky to get into as a non-expert or novice GALE developer. On top of that, repeatedly defining adaptive behavior is tedious and typically involves a lot of repetitive work. Authoring tools are used in order to solve these problems and to provide support when authoring GALE applications. The tool which is currently used for authoring applications in GALE is called GAT (GRAPPLE Authoring Tool). This tool, in its current state, does not contain the expected and required functionalities. The reason for this is that GAT has been developed by a number of different parties, leading to some unfortunate design decisions and usability issues. The solution to this problem is to design and create a new authoring tool for GALE.

A new authoring environment for GALE called ALAT is presented in this thesis. It has a low entry barrier and the usability for novice GALE users is high. It is a generic and extensible platform which can be used for numerous adaptive hypermedia application types. A target audience and potential application area is determined for ALAT as well. We use this target audience and application to compare it to other cutting edge authoring environments created over the past few years. This comparison shows that ALAT holds up very well with regards to its target audience and application area.

List of Abbreviations

AEH	Adaptive Educational Hypermedia
AES	Adaptive Educational Systems
AHA!	Adaptive Hypermedia for All!
AHS	Adaptive Hypermedia Systems
ALAT	Adaptive Learning Authoring Tool
GALE	Generic Adaptation Language & Engine
GAM	Gale Adaptation Model
GAT	GRAPPLE Authoring Tool
GRAPPLE	Generic Responsive Adaptive Personalized Learning Environment
HTI	Human Technology Interaction
HTML	Hyper Text Markup Language
ITE	Intelligent Tutoring Environment
JSON	JavaScript Object Notation
MOT	My Online Teacher
MS	Microsoft
PRT	Pedagogical Relation Tool
TU/e	Eindhoven University of Technology, the Netherlands
UX	User Experience
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

Table of figures

Figure 1.1: A snippet from the Milkyway demo 2
Figure 2.1: The Interbook Interface 6
Figure 2.2: The AHA! Graph Author 7
Figure 2.3: The GAT Architecture 8
Figure 2.4: The GAT Domain Designer 9
Figure 2.5: The GAT Course Designer 10
Figure 2.6: The GAT PRT 11
Figure 2.7: The GAT Welcome Interface 13
Figure 2.8: A screenshot of ACTSim. Taken from [GaDa10] 15
Figure 2.9: The WOTAN directed graph view on the left and indented list view on the right. Taken from [FrRo05] 17
Figure 2.10: A screenshot of the PEAL editor. Taken from [CrSm09] 18
Figure 2.11: A snippet of the MOT 3.0 web interface 18
Figure 2.12: A screenshot of the AMAS interface. Taken from [GaCo14] 19
Figure 3.1: The step by step screen 22
Figure 3.2: The overview screen 23
Figure 3.3: The rule target selection controls 23
Figure 3.4: The settings screen 24
Figure 3.5: An example blueprints template 27
Figure 3.6: An example pedagogical relation definitions template file 29
Figure 3.7: The ALAT client-side Application Architecture 31
Figure 3.8: The ALAT Back-end Architecture 31
Figure 3.9: An ALAT project set to be only partially deployed 32
Figure A.1: The login screen 43
Figure A.2: The register screen 43
Figure A.3: The project selection screen 43
Figure A.4: Adding a concept in overview mode 44
Figure A.5: Adding a concept in the step-by-step view 44
Figure A.6: Adding a unary rule 45
Figure A.7: Adding a binary rule 45
Figure A.8: Adding an attribute 45
Figure A.9: The advanced settings screen 46
Figure B.1: The step-by-step design 48
Figure B.2: The Overview design 48
Figure B.3: Adding a new item in the step-by-step view 49
Figure B.4: The settings screen design: 49
Figure B.5: Adding a prerequisite in the settings screen 50
Figure C.1: The ALAT front-end architecture 52

Table of contents

1 An introduction to adaptivity 1
1.1 GALE 1
1.2 Adaptivity in GALE 2
1.3 The structure of GALE applications 2
1.4 Authoring as a project goal 3
2 Related Work 5
2.1 Analysis of the past and present 5
2.1.1 Previous authoring tools 5
2.1.2 The present situation 8
2.1.3 Discussion 13
2.2 A comparative study 15
2.2.1 ACTSim 16
2.2.2 WOTAN 16
2.2.3 MOT 17
2.2.4 AMAS 19
2.2.5 Discussion 20
3 ALAT 21
3.1 The ALAT authoring interface 21
3.2 Templating 25
3.2.1 Format and structure 25
3.2.2 In-application templating versus file editing 25
3.2.3 Blueprints 26
3.2.4 Relation Definitions 28
3.3 The ALAT system architecture 30
3.3 Generating GALE applications 32
3.3.1 Generating a gam file 32
3.3.2 Deploying GALE applications to the server 33
4 Target audience and applications 34
4.1 The ALAT target audience and applications 34
4.2 Possible future applications 34
5 Conclusion 36
6 Future Work 37
6.1 Improvements & Additions 37

6.2 Research & Development 37
Bibliography 39
Appendixes 41
Appendix A: The ALAT User Interface 42
Appendix B: The ALAT User Interface Design 47
Appendix C: ALAT Architecture (Elaborated) 51

Chapter 1

An introduction to adaptivity

Many items we use on a daily basis are tailored to our personal needs and desires. Furniture and clothing are examples of this: The height of many office chairs is adaptable, for example and a tall person needs bigger clothes as opposed to a shorter person. Over the years this phenomenon has made its way into the digital world as well. The reason for this is that people have their own preferred learning style, their own interests and their own approach to problem solving. In the past few years especially, personalization and adaptation has shown up in fields such as advertising, shopping recommendations and on-demand television. Even though the digital applications of adaptivity have grown over the years there are still some areas in which adaptive hypermedia could make a big difference. The area of e-learning is a prime example of this, though the institutions are slowly realizing that a one-size-fits-all program is not the ideal way of teaching.

The field of adaptive hypermedia systems (AHS) focuses on developing platforms and techniques which further explore and exploit the possibilities in digital personalization and adaptation. It does this by adding adaptive behavior to hypermedia content. The term hypermedia is an encapsulating term which covers audio, video, plain text, hyperlinks and other visual media. In this case hypermedia is typically viewed in an internet browser. The adaptive behavior is based on environmental factors as well as the user browsing behavior and history.

The concept of adaptive hypermedia was first elaborately described by [BrLe93] in 1993. Soon after, pioneering ITEs (Intelligent Tutoring Environment) containing adaptation functionalities such as ELM-ART [BrEl96] were created. Since then a multitude of adaptive hypermedia platforms have been developed over the years. The adaptive hypermedia platform this thesis covers is called GALE. GALE is the AHS used for research and teaching at TU/e[1].

1.1 GALE

GALE [SmBr11] (Generic Adaptation Language & Engine) is a strong generic platform for adaptation to individual users. It acts as a web server in that it serves web content to the client, but it also handles adaptation rules and expressions. Almost every adaptation technique can be applied in GALE, which makes it a truly generic platform.

GALE has been developed as a part of the GRAPPLE [BrAl13] (Generic Responsive Adaptive Personalized Learning Environment) project. This is a EU FP7 STREP[2] project that lasted from 2008 until 2011. In this project academic and industrial partners collaborated to develop a generic adaptive learning environment and to integrate this with several learning management systems.

[1] Eindhoven University of Technology, the Netherlands
[2] Specific Targeted REsearch Project
(http://cordis.europa.eu/fp7/ict/future-networks/funding-schemes_en.html)

An input is required to define the adaptive behavior GALE has to execute. Various formats and models are supported by GALE, but the format most used in this thesis is GAM (Gale Adaptation Model). This input is responsible for all adaptive behavior in the resulting adaptive application. More about GAM and how it is applied can be found in *section 1.3*.

1.2 Adaptivity in GALE

The main three types of adaptation found in GALE are content adaptation, navigation adaptation and presentation adaptation. Content adaptation is used to personalize the content of hypermedia pages based on the user. This is done by extracting information from a user model, which contains information about the current user.
Navigation adaptation involves the adaptation on the availability of hyperlinks, thus influencing the users' path of navigation through the available hypermedia pages. Lastly, presentation adaptation changes the way the page information is displayed rather than the information itself. This could result in different text sizes or fonts, but it could also go as far as to display a different kind of media to represent information altogether (video instead of text, for example).

Content should first be created in order to be presented through GALE. The format in which content needs to be presented to GALE is HTML combined with some specific GALE tags for further customization. These HTML pages are linked to parts of a domain and adaptation model which also have to be presented to GALE. Both of these models are represented by a single GAM file. This file is written in a textual format which is aimed to be readable by humans and which describes a domain and its associated adaptation. However, it is difficult for non-expert users to create this file without extensive knowledge on GALE. Designing and creating such a GAM file containing the adaptive application content is also known as "authoring".

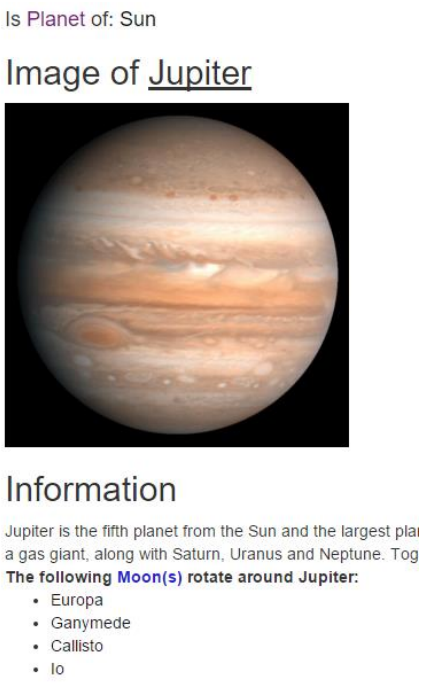

Figure 1.1: A snippet from the Milkyway demo

1.3 The structure of GALE applications

All concepts in a GALE application are contained within their own separate entities. These entities consist out of a set of attributes and parameters, amongst which are the concept name and the web resource (used when the concept is accessed and displayed within the browser). These concepts can be connected to each other by means of labeled relations. Many standard layouts for GALE, such as the "*static-tree-view"* [SmBr11] make use of parent-child relations in order to create a default concept hierarchy. These labeled relations can also be used to create dynamic behavior on concept pages by listing all objects that are related by the current object through a specific labeled connection. An example of this can be found in the Milkyway demo. In

this demo, an "*isMoonOf*" relation is used to list all moons rotating around a specific planet. A snippet of the concept "Jupiter" with such a list is displayed in *figure 1.1*. These relations do not have any specific adaptive behavior attached to them and are often referred to as "non-pedagogical relations".
The set of all concepts and the relations they share will be referred to as the domain model.

Because the domain model is only responsible for the structure and data of the GALE application a second model is necessary in order to define the adaptation. This is called the adaptation model. This model has been called the "*Achilles heel of adaptive application design*" [BrSm12] due to the increase in complexity as more flexibility in adaptivity is introduced. This model consists out of a set of adaptation rules which are responsible for the adaptive behavior of the resulting GALE application. These adaptation rules can also involve multiple concepts, forming pedagogical relations.

These models are brought together in the resulting GAM file. In this file all concepts are declared together with their specific attributes, parameters and GAM-expressions representing the adaptive behavior defined in the adaptation model. This file, named "*concepts.gam"* can be deployed on the GALE-server where it will then be automatically detected and deployed by GALE.

An authoring environment is used to create domain and adaptation models and deploy these into GALE applications. The goal of this is to make the authoring process less complex while trying to retain the flexibility that GALE allows.

1.4 Authoring as a project goal

Authoring has been a difficult subject throughout the history of adaptive hypermedia. A wide variety of domain and adaptation model design techniques has been created in an attempt to make this task easier. This also holds for GALE.

The current authoring environment for GALE is called GAT. It has a graphic user interface which can be used to author adaptive applications. GAT tries to make the authoring process less complicated by visualizing the domain and adaptation model to an extent at which it is easy to create, understand and edit. However, time and use have proven that GAT does not have the desired functionality and usability of an authoring environment for GALE. An in-depth analysis of GAT and other related work is presented in *chapter 2*.

This calls for a new authoring environment. Lessons learned from GAT and other earlier projects combined with fresh ideas can be used to create a new and modern tool.

In order to get new ideas and inspiration the company called "De Roode Kikker" has been involved in this project. "De Roode Kikker" is a company which created and maintains an educational platform. They are looking into the possibility of using GALE in order to make their educational content adaptive. That means their goal is to create a specific platform which is aimed at authoring content within their existing platform. Their interest lies in adaptivity focused on the area of Adaptive Educational Systems (AES).

The new authoring environment desired for GALE should be a generic platform with which a multitude of adaptive hypermedia types can be authored.

Thus, there is a discrepancy in wishes and requirements between the two parties involved (TU/e and “De Roode Kikker”). Yet, this does not hinder the collaboration in terms of exchanging inspiration and ideas as well as providing mutual technical support.

Therefore, the goal of this project is to create a generic platform which helps content authors to easily create adaptive hypermedia without extensive knowledge of GAM or the inner workings of GALE. In the ideal authoring environment the user will not have to deal with any kind of code at all. This platform needs to be generic in the sense that the adaptation must be adaptable and extendable. The material authored by the authoring environment must be field-independent. As a result, ALAT has been created. ALAT is a new authoring environment for GALE

Chapter 2

Related Work

This chapter provides a connection between ALAT and other authoring environments created for various adaptive hypermedia engines. This is done by first analyzing the past and present situation, analyzing GAT and its preceding authoring tools. The analysis of GAT will be especially detailed, as this is the current tool for GALE.
This analysis is then followed by an analysis of other more recent projects, in an attempt to draw an image of ALAT's relevance in the field.

2.1 Analysis of the past and present

A multitude of authoring tools has preceded the new GALE authoring environment. This section will give an analysis of some established authoring tools present in the field of adaptive hypermedia up until GAT, which is the current authoring environment for GALE. The preceding tools will be presented first, followed by a more extensive analysis on GAT. The goal of this analysis is to get a clear view of what has been developed in the past. Further design decisions made in the development of the tool are justified based on these past developments. The analysis will include a description of the chosen tools together with their main features. This information will then be used to discuss the main strengths and weaknesses of the tools.

2.1.1 Previous authoring tools

When looking back at the beginning of adaptive hypermedia research, the first extensive description of a published authoring tool was on Interbook [BrSc96] back in 1996. Interbook is a system used in the authoring and deployment of adaptive electronic textbooks. An example of an Interbook application can be seen in *figure 2.*1. The right panel represents a glossary of the current text book. The left panel shows the concept content in the bottom left panel. The top left shows the position of the current concept in the textbook. The panel on the right shows both the background and outcome concepts. A background concept serves as a prerequisite. The link directs the user to a concept which helps to understand the current concept. The outcome concepts represent parts of the knowledge presented in the current concept. Both of these types of adaptation rules are defined by the annotations made during the authoring of this electronic textbook.

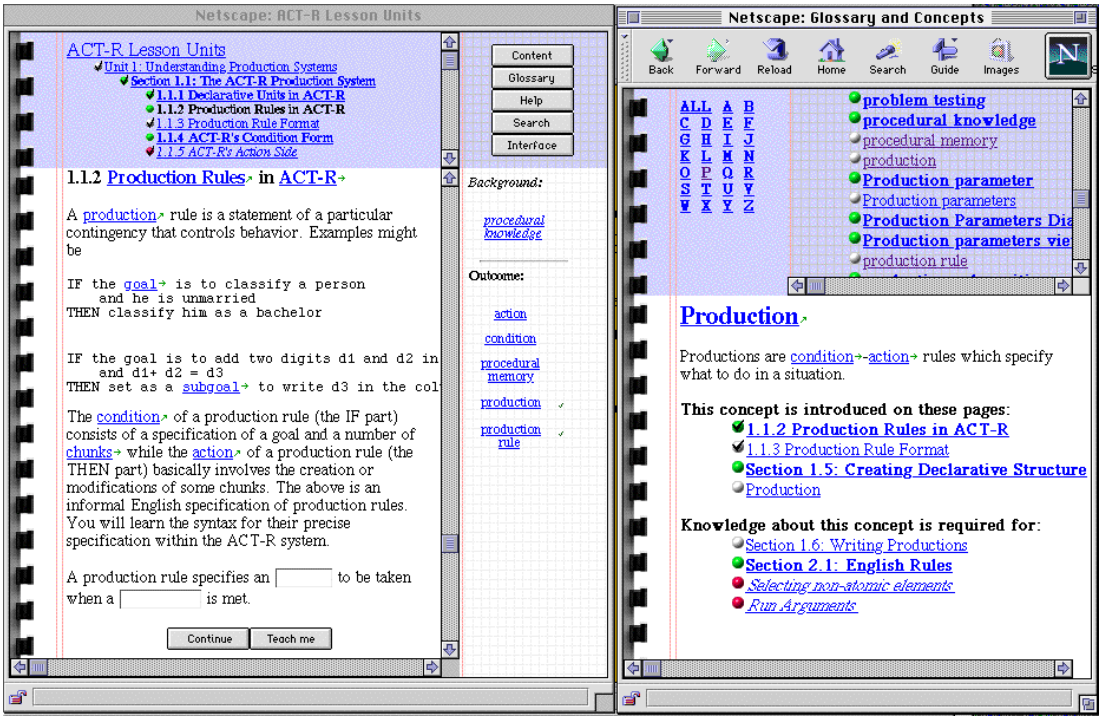

Figure 2.1: The Interbook Interface

Authoring in Interbook is done by importing all content text in Microsoft Word and annotating it. By annotating the author defines both the domain as well as the adaptation model. This means the author can author domain, content and rules (background/outcome concepts) in a single process. This annotated Word document can then be converted to HTML and be served by the Interbook server.

A great advantage of this early form of Authoring lies in its simplicity. Using MS Word as an authoring environment creates a sense of familiarity. This lowers the entry barrier and helps to get the author started quickly.
Even though this first form of authoring is easy to learn and has a familiar environment to author in, it does have some drawbacks that make it difficult to use in more modern authoring systems.

The first drawback is the lack of support from the authoring environment for creating “sensible” or at least sound adaptation models. All concepts and rules have to be managed manually and are only textually defined in the Word document using annotations. Secondly the number of types of adaptation rules available is limited and there is no way to expand this set. On top of that the number of applications of Interbook is limited to adaptive electronic textbooks and the authored content is limited to MS Word content. This makes it very difficult to use Interbook to create more modern adaptive hypermedia applications.
Many projects developed after Interbook have tried to tackle these issues in order to create a more generic and extendable platform.

A new set of authoring tools emerged with the development of AHA! [BrSm06]. AHA! is a general-purpose adaptive hypermedia platform which aims to be as generic as possible. Its development started in 1996 and lasted until about 2007 (right until the GRAPPLE project started. See *chapter 1*). It is also a direct predecessor of GALE, for which the authoring tools will

be discussed later this chapter. The main authoring tools developed for AHA! are the Concept editor and Graph author[3]. Authoring for AHA! is very different from the authoring process described for Interbook as the content authoring is separate from authoring the domain and adaptation models. The possibilities and flexibility in content creation have improved greatly by using HTML/XHTML-content instead of MS Word content.

The concept editor uses an indented list structure to create a model and a properties screen to set rules and conditions on the created structure. This editor relies on pseudo-code in order for the user to set up the course. This puts the user really close to the actual domain/adaptation model code and requires the user to have knowledge of coding on top of adaptive hypermedia authoring. The graph author is recommended over the concept editor for most users. The main reason for that being that the graph author helps the user with a lot of technical authoring complexity. The graph author solves the problem of having to deal directly with the AHA! domain/adaptation model code by using templates. This moves all the AHA! adaptation code under the hood, so that the user does not have to deal with it. Both the adaptation rules as well as the standard concept structure is stored in these templates.

When looking at the graph author interface in *figure 2.2*, there is an indented tree structure which shows the hierarchical topic structure in the shape of parent-child relationships on one side and a graphic representation of the course and its relations on the other. The hierarchical structure is used to represent the domain model and has the shape of an indented view in the resulting AHA! application. The right side of the screen shows the user all pedagogical relations between different course concepts in the form of a graph. This is a visual representation of the adaptation model. The adaptation rules represented by these relations are drawn from a

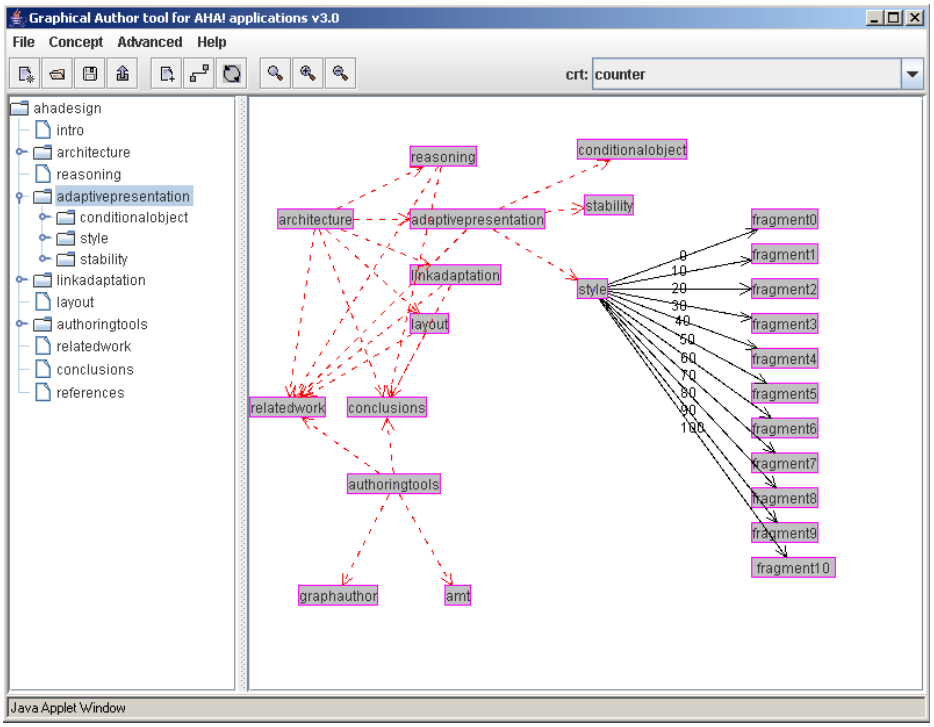

Figure 2.2: The AHA! Graph Author

[3] AHA! Also has a Form editor but we will not discuss this special-purpose tool.

template and are thus dynamic. This results in rules which are reusable and easy to implement. This is a strong feature in terms of usability, extendibility and maintainability.

However, it can already be seen in *figure 2.2* that the graph interface which holds all pedagogical rules and additional relations becomes cluttered quite quickly, making it unclear as to which relations have been applied. Another issue is the declaration of additional concept attributes. This cannot be done using the graph author. All attributes and parameters used have to be declared in the concept templates.

Even though the graph author helps the user to author content without the use of extensive knowledge, there is room for improvement. The GRAPPLE project has been an opportunity to create a successor of AHA! and its authoring environment .

2.1.2 The present situation

The graph author already separates the authoring of the domain and adaptation model. This separation has become even greater in GAT. Whereas the graph author still has the display of a hierarchical structure and the course rules and conditions in one screen, GAT has completely separated these into a course and a domain designer.

GAT has been developed as part of the GRAPPLE project as described in *chapter 1*.

This has resulted in a distribution of tasks in which multiple parties created different aspects of GAT. The separation of tasks amongst different parties has caused further separation between different aspects of content authoring. The authoring of domain and adaptation model has separated to a point at which they are created in separate tools. There is a tool for creating domains, a tool for creating an adaptation model and a tool for designing new pedagogical relationship types. These tools are implemented in the GAT shell which runs within the browser screen as depicted in *figure 2.3*. The DM Tool represents the Domain designer, the CRT tool

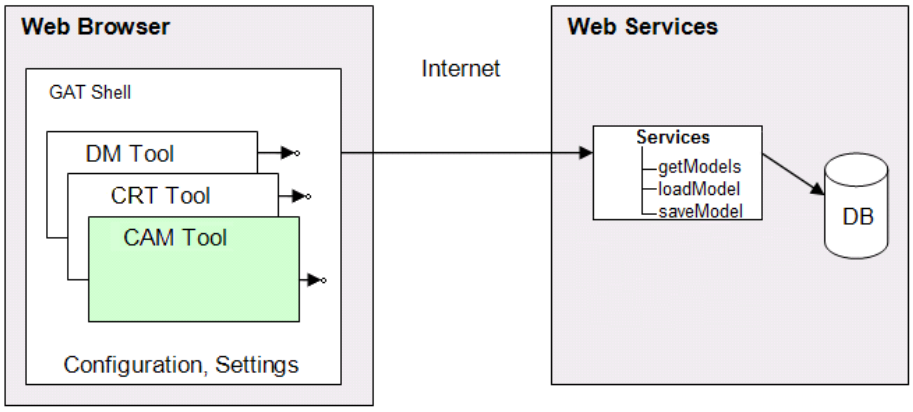

Figure 2.3: The GAT Architecture

represents the Pedagogical Relation Tool (PRT) and the CAM tool represents the Course designer tool. As in the AHA! authoring process, the authoring of content is separate from the domain and adaptation model authoring. This application then depends on a back-end to provide and store project data. Because the tools are created by different developers, there have been a number of unfortunate consistency issues and disconnects in workflow between these different tools (tool switching to perform minor tasks, for example).

The Domain designer tool

The designing of an adaptive application usually starts with defining a domain. In GAT this is done using the domain tool. This tool enables the user to build domains. Unlike the graph author, this editor uses a graph-like structure in which any non-pedagogical relation can be defined. The nodes represent concepts and the edges are the relations between these concepts. These relations are labeled connections between concepts, which can later be utilized to apply rules and conditions on or be used at page level to create lists and links. This means that all relations the author wants to use in his course have to be defined before starting to build the adaptation model. When an author decides a new concept should be introduced while working in the course designer, he has to move back to the domain designer in order to create this concept. After adding this new concept the user has to return to the course designer and reload the domain model.

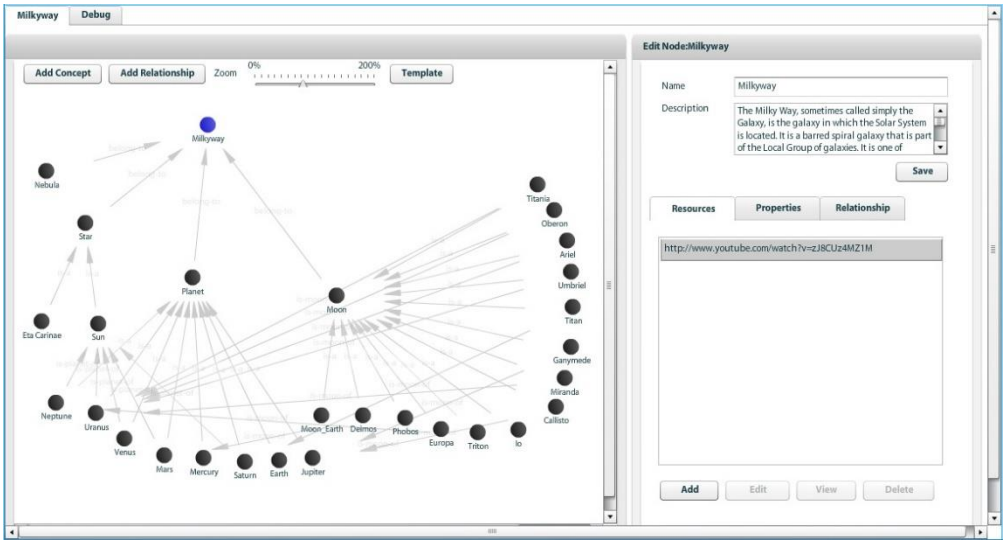

Figure 2.4: The GAT Domain Designer

The models created in this tool have a graph structure in which the user can add or delete nodes and relations. Unlike the nodes created in the graph author, each node and relation in these editors also has their own set of properties which the user can edit.

All main desired functionalities are available in this editor in some shape or form, yet there is room for improvement and change. Domain models usually consist out of a big connected graph, which can grow up to dozens or hundreds of nodes depending on the domain which is being modelled. Whereas the graph author of AHA! had a clear indented list of the current domain, the graph interface of the domain tool in GAT makes it difficult to see which nodes are root nodes, for example. The possibility to zoom and pan the canvas in which this graph resides makes it even more difficult to get a good view of how everything is structured. It is even possible to entirely lose unconnected nodes located on the edges of the canvas. As can be seen in *figure 2.4*, the interface becomes cluttered as the domain model grows.

Adding a new relation to the domain model involves a separate menu with selection boxes which contain a list with all candidate source and target concepts. When designing a large domain, this list grows to an unmanageable size.

A goal of the redesign of the domain designer is a system in which it is easier to keep track of individual nodes and their connections. It will also make it possible to rework functionalities which are currently divided amongst several menus into a one-screen solution.

The course designer tool

After a domain has been created, the adaptation model has to be constructed using the course design tool. In GAT, the adaptation model is called a “course model”. This course model is a set of adaptation rules and conditions applied on subjects of one or multiple domains. This determines the behavior and adaptivity of the course. As in the graph author, the code of these rules is not visible to the user.

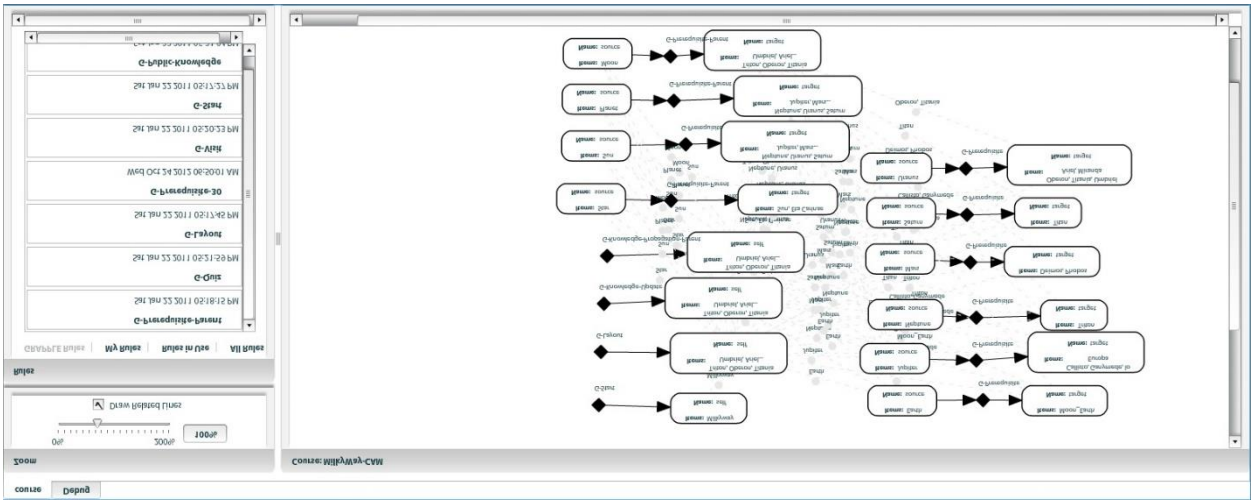

Figure 2.5: The GAT Course Designer

The course designer also uses the same drag and drop graph-like visual interface as the domain designer. It takes domain concepts and introduces rules and constraints on them such as prerequisite constraints, knowledge propagation rules or starting topic rules. GAT provides a predefined set of rules, created using PRT described below. One must use PRT in order to create custom rules and constraints. After restarting the course designer these new rules become available. A fully designed course usually takes the shape of a forest of single nodes, pairs of connected nodes or connected sets of nodes (*figure 2.5*).

During the GRAPPLE project, a “standard” set of 15 rules was created for GAT. While it might seem practical to have a rule for each conceivable situation, even out of this set of just 15 rules, most rules are hardly ever used. A few rules are used so often they could be called “standard behavior”. It makes little sense to bother the user with setting up these rules for every concept as they apply to the entire current course design.

The rules defined here can be set up using “sockets”, which are represented by nodes within the working canvas. Depending on the type of relation (unary or binary) these sockets are connected to each other with a single edge. This represents the source and target of pedagogical relations. The sockets, unlike the domain tool, can have multiple concepts connected to them. This makes the interface much less cluttered when compared to the graph author as multiple relations can be fit into a single socket. On the other hand it is difficult to see which concepts are involved in these adaptation rule sockets and it is difficult to get an overview

of which relations any one concept is involved in. Another big problem arises when selecting input for the sockets in the course designer. When a domain grows to dozens or hundreds of concepts, the list of topics the user can choose from often becomes too long. This is the same issue as described in the previous chapter regarding setting up new relations in the domain designer. This makes it hard to easily choose the correct items out of this list. This is a serious usability problem, as it already occurs in a basic example course provided in GALE (the Milkyway course).

The pedagogical relation tool

The third tool in GAT aids the more advanced user when creating new pedagogical rules and relations. This is very useful because it gives the possibility to replace or extend GAT's standard set of rules. Various functionalities and attributes can be defined in this tool as well as settings concerning the visual representation of this rule in the course designer (*figure 2.6)*.

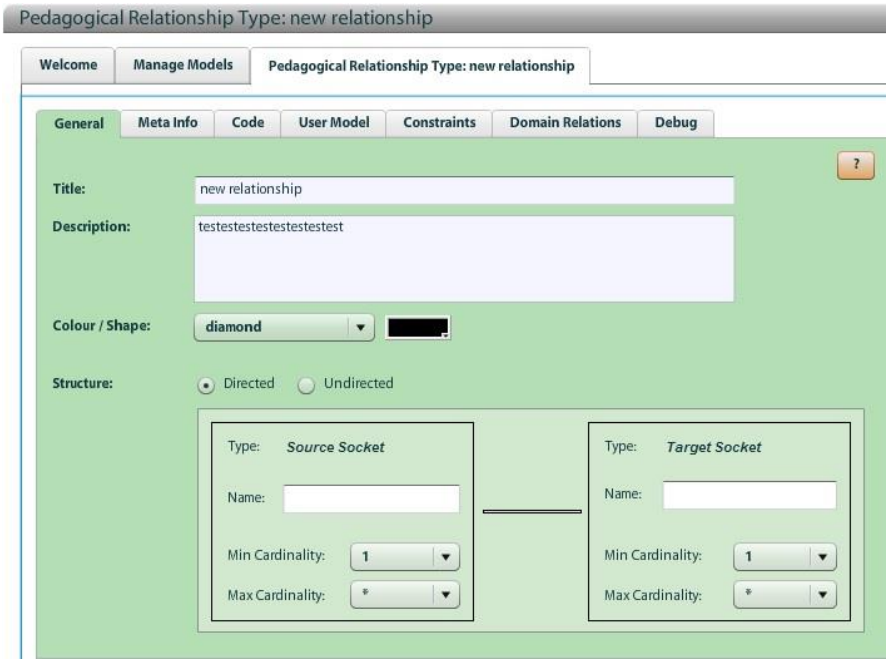

Figure 2.6: The GAT PRT

The underlying functionality of using concept and rule templates already existed in the graph author, but an interface for creating these templates makes it easier to add new adaptation rules without having to edit any configuration files. Dynamic behavior is a strong feature in terms of maintainability and extendibility. Another smart design choice is to "hide" this tool under the advanced mode. This kind of feature has to be considered as well when designing a new authoring environment. Being able to extend the tool behavior and application benefits its usability and gives it a wider field of application.

A toolset divided

The decision to separate the authoring process into multiple tools has made it difficult for new users to understand the connection between the domain and adaptation model. By having the user create rules for which he has to select the source and target subject instead of having the rules being selectable per concept, the user has to start his thinking process at the rule rather than the concept. i.e. The user selects a rule and selects what concepts to apply it on instead of first selecting a concept and picking the rules that should apply to it. This is unintuitive and makes it more difficult, especially for a new user, to create an entire course (that is; both a domain model and an adaptation model).

These problems, together with the need of context switching between course and domain designer give reason to create a new tool. As will be shown in *chapter 3*, the two workflows can be molded back into a single, more intuitive workflow.

Thus, when regarding the GAT-workflow, it can be concluded that the separation of the adaptive course authoring process into a domain and course model makes it more difficult and tedious to author adaptive applications. The templating and reusability of pre-constructed rules and conditions which can be applied to course subjects however, make the authoring tool more user-friendly and greatly lowers the entry barrier. An engineer will be needed to set these templates up, but their application is fairly easy and can be taught to a new user with ease.

Techniques

Apart from issues with the workflow and usability of GAT, there are some aspects that also hinder its overall quality to some extent.

Firstly the application has a multitude of minor bugs and visual glitches in all tools. This causes several issues which justify the use of a different technique for the creation of a new authoring tool. This involves issues like drop-down menus not working and the flickering of the screen. The application also has some serious performance issues when building large models or when using it for extended periods of time. This makes the domain and course designers quite difficult to use in larger projects.

When a domain or course is saved, or when an error occurs, the application feedback is delayed for a few seconds at times. It is difficult to determine whether this is due to some client-side process or the application back-end. These are some of the main factors that harm the application responsiveness and usability.

Another point of critique is that the user interface overall is not up to web standards. Many aspects of the design do not hold up in terms of visual design and menu navigation. On top of that, the overall interface is cluttered and unclear (*figure 2.7)*.

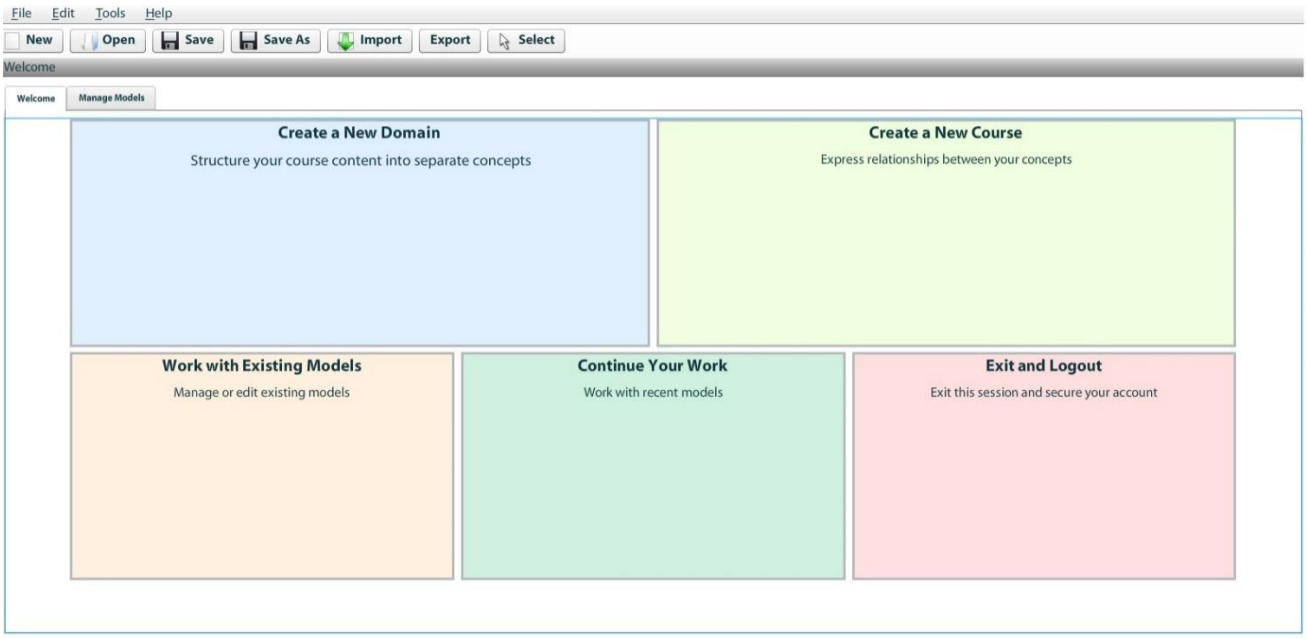

Figure 2.7: The GAT Welcome Interface

When creating a web-based application it is important to keep the interfaces easy to understand and streamlined. A redesign of the user interface is very much justified. Stripping down the number of controls, combined with better user guidance will make the new authoring tool much easier to understand and learn. This can all be done without removing any important features from the authoring tool.

2.1.3 Discussion

When looking at GAT and its predecessors, it becomes apparent that there are a lot of lessons to be learned from these authoring environments.
For example, the separation of content authoring from the domain and adaptation model as seen after Interbook is worth preserving as the creation of modern hypermedia content, often written in HTML code, is supported by numerous editors and is very well documented.
Interbook has a familiar and easy way of authoring, but the lack of in-application support, a restricted application domain and the lack of adaptation rule diversity hold back the authoring of more complicated domains.

AHA! as a predecessor of GALE has a more diverse authoring environment.
The graph author provides a clear view of the domain model and templating adaptation rules contributes to a dynamic extendable application behavior. However, concept customization is limited as all attributes and parameters are templated or set at creation time, leaving out the option to edit or insert new values. Lastly, the cluttering of the adaptation rule interface is a big problem in larger adaptation models.

The analysis of GAT reveals that it has a wide range of functionality as well as the possibility to extend and add new adaptation rules. The separation of the authoring process into multiple tools puts some unfortunate constraints on the application responsiveness, maintainability and accessibility. A part of the extensive templating the AHA! graph author had has been lost, but

GAT does allow more extensive customization of its concepts in terms of properties and attributes as well as the creation of non-pedagogical relations on the fly.

Even though the domain designer makes introducing non-pedagogical relations very easy and dynamic and the course designer makes introducing adaptation rules and relations much less cluttered, the context switching and interface cluttering still hold back the user-friendliness and workflow fluency.

The strong features and restrictions of each of these tools creates a guideline and a vision which is leading in the creation of a new authoring environment for GALE. A clear interface has been a problem for almost all previous tools. Either the domain model or the adaptation model becomes cluttered or difficult to keep track of in all of these tools. Keeping track of which rules and relations are associated with a particular concept is crucial in the authoring of an adaptive application. A middle ground should be found between the clear hierarchical views of the graph author, the diversity in domain and adaptation model design in GAT and the clear and simple rule annotation listings of Interbook.

Templating and the possibility to extend templated concepts and relations is another important feature. It should be possible to incorporate both the extensive concept templating done in AHA!, as well as the concept customization options GAT has.

The resulting new authoring environment should be a modern web-based application with a revised implementation of all features described in this chapter. It should have a user-friendly non-cluttered interface and workflow, allowing the authoring of advanced adaptive applications with ease and a low entry barrier.

2.2 A comparative study

Many different adaptive hypermedia engines have been developed over the last two decades. In this chapter ALAT will be compared to several authoring tools of these engines. These tools have been selected in order to point out interesting comparisons in design philosophy and functionality. This comparison also provides an indication as to the relevance of ALAT in the field of adaptive hypermedia. The comparisons are followed by a discussion on major differences in functionality and design philosophy. Various points made in this discussion will be used to contribute to *chapter 7*, which deals with possible future work regarding ALAT. Some key features of ALAT are mentioned during the comparisons between different systems. A more in-depth analysis of ALAT and the features mentioned in this section will be given in *chapter 3*.

Many early adaptive hypermedia engines provided only limited support for authoring. Prime examples of this are Interbook, as discussed in chapter 2 and KBS Hyperbook [FrNe98]. Interbook used MS Word for authoring and KBS Hyperbook relies on the creation of XML files to create adaptive applications [QuNe02]. These early forms of authoring do not provide a lot of user support and require the user to provide either annotations or adaptation code manually. More modern systems such as AHA! use extensive user models and have a separation between authoring adaptation and authoring content and presentation. This allows the system to manage nearly arbitrary adaptation rules and arbitrary presentation [Sm11]. The following comparisons have been made on dedicated authoring environments which author domain and adaptation models rather than content and presentation.

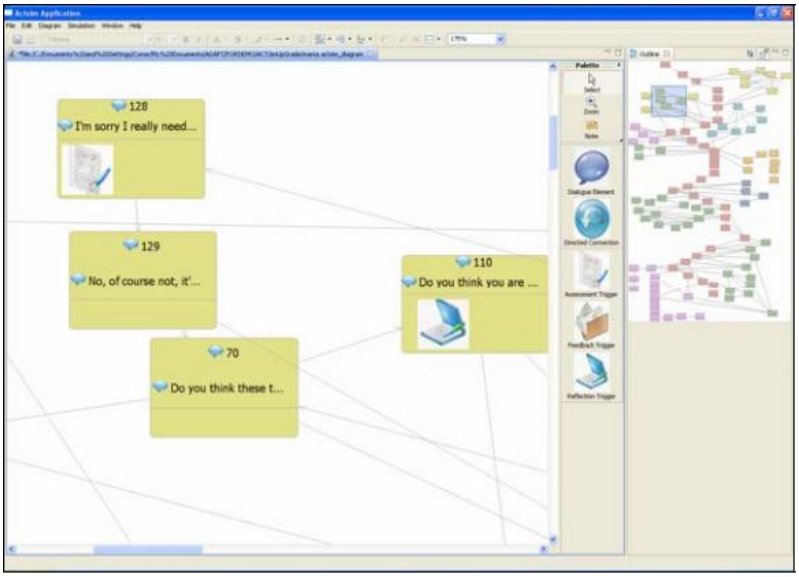

Figure 2.8: A screenshot of ACTSim. Taken from [GaDa10]

2.2.1 ACTSim

Apart from hypermedia engines that support almost arbitrary adaptation rules, there also exist modern systems which do not focus on this notion. An example of such a system is ACTSim [GaDa10]. ACTSim is a “unique composition tool that supports the rapid development of personalized training simulation”. Its focus is making educational soft skill simulations adaptive. The creators of ACTSim claim these simulations are difficult to author as is, yet ACTSIM provides enough support for a non-technical expert to create simulations with ease (*figure 2.8*). When comparing ACTSim to ALAT it becomes apparent that the scope of adaptive hypermedia that can be authored with this tool is fairly limited. This limitation however, is a double edged sword. Because of the limitations in authoring scope, ACTSim can be used to author educational simulations to a much further extent than can be achieved by generic tools such as ALAT. When creating an authoring tool a tradeoff has to be made between specialization and genericity. As mentioned earlier in this thesis, ALAT is aimed to be as generic as possible. The comparison with ACTSim shows another possible design philosophy with regards to specialized adaptive hypermedia authoring tools as opposed to more generic tools

2.2.2 WOTAN

WOTAN [FrRo05] has tools that take an interesting approach in user interface design. It has tools for both an indented list hierarchy (as in ALAT) as well as an interface representing the current project in a directed graph (as in GAT). This, in essence, is a strong set of views. The downside of this representation is again the graph implementation. Though some visualization techniques such as automating the layout and clustering groups of nodes are added to prevent cluttering, larger projects still suffer from this graph representation in terms of complication. This is caused by the many different types of nodes within this graph representation. These visualization techniques make the graph a lot more complicated. Another feature of WOTAN that goes against the principles of ALAT is the representation of adaptation rules. These rules are represented as items in the project hierarchy or as nodes in the graph representation as can be seen in *figure 2.9*. This causes a conflict of interest in the adaptive course design as both the domain as well as the adaptation model are overlapping in the main project structure interface. The interface also becomes more confusing when concepts with multiple parent concepts are introduced. The result of this is that these concepts appear multiple times in the indented list project hierarchy as indicated in *figure 2.9*. This is why ALAT does not allow these kind of domain connections (and uses a tree structure instead).

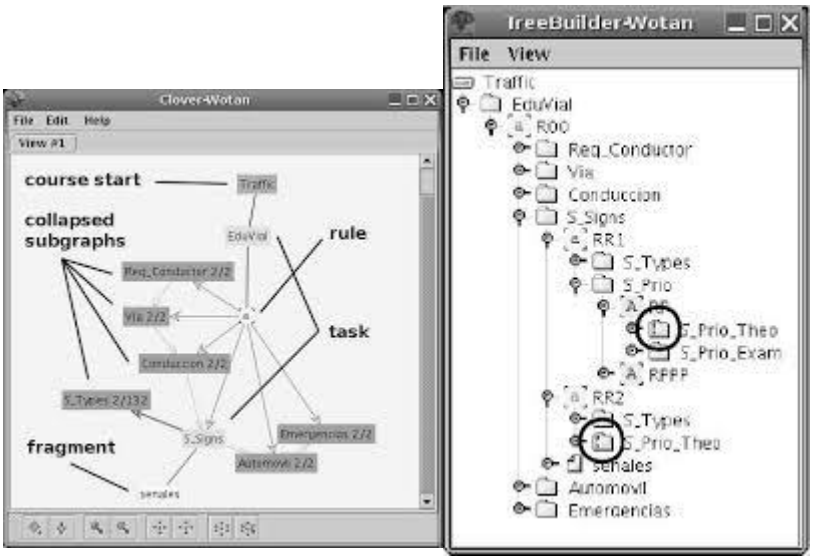

Figure 2.9: The WOTAN directed graph view on the left and indented list view on the right. Taken from [FrRo05]

2.2.3 MOT

Whereas WOTAN had some issues regarding separation of concerns, MOT[4] 3.0 [FoCr10] focusses on consistency regarding this subject. It does this by closely following the five-layer LAOS framework [CrMo03]. The corresponding layers are: Domain model, Goal model, User model, Adaptation model and Presentation model[5]. MOT is a web-based authoring system used for online adaptive course production. It is used in combination with the PEAL adaptation strategy author to author adaptive hypermedia applications. It relies on other adaptive hypermedia engines such as AHA! to deliver its courses [CrSm05].

Because of the extensive layering and separation of concerns in MOT, its authoring process is more modular as opposed to the one-stop-shop process in ALAT. Domain models are constructed in a dedicated web-interface and the adaptation model is created using the PEAL adaptation strategy author. This is an application which supports the user in writing the adaptation code by implementing status bar suggestions to improve code validity, code completion as well as a strategy wizard to define and initialize variables. Because of the separation of concerns, attributes that should be stored in the user model are declared separately from the ones that should not (the former are declared in the web interface, the latter in PEAL).

The main differences in authoring support between MOT and ALAT can be partially explained when comparing their respective design philosophies. MOT is created such that content authors can set up a domain using the MOT web interface (*figure 2.10)*. Adaptation authors then use PEAL to create the appropriate adaptation strategy (*figure 2.11*). The adaptation language used in PEAL is called LAG [CrSm09]. ALAT uses the role of an adaptation expert to set up templates,

[4] My Online Teacher
[5] This thesis does not elaborate on this model. [CrMo03] provides an in-depth analysis.

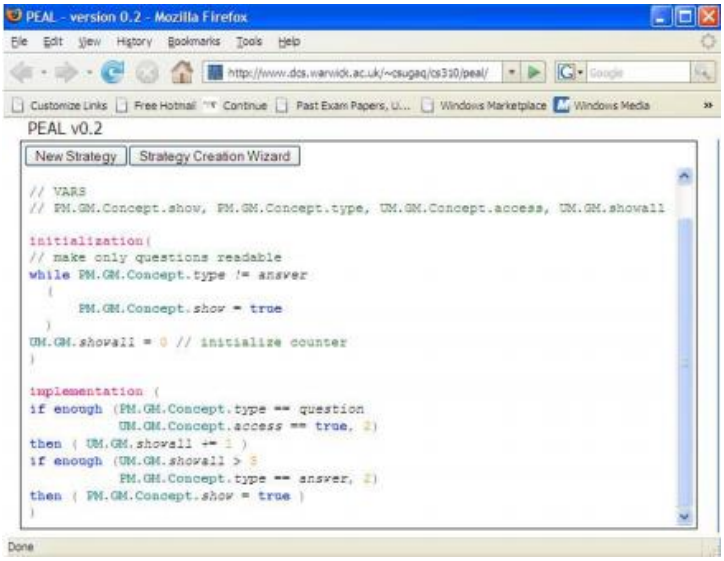

Figure 2.10: A screenshot of the PEAL editor. Taken from [CrSm09]

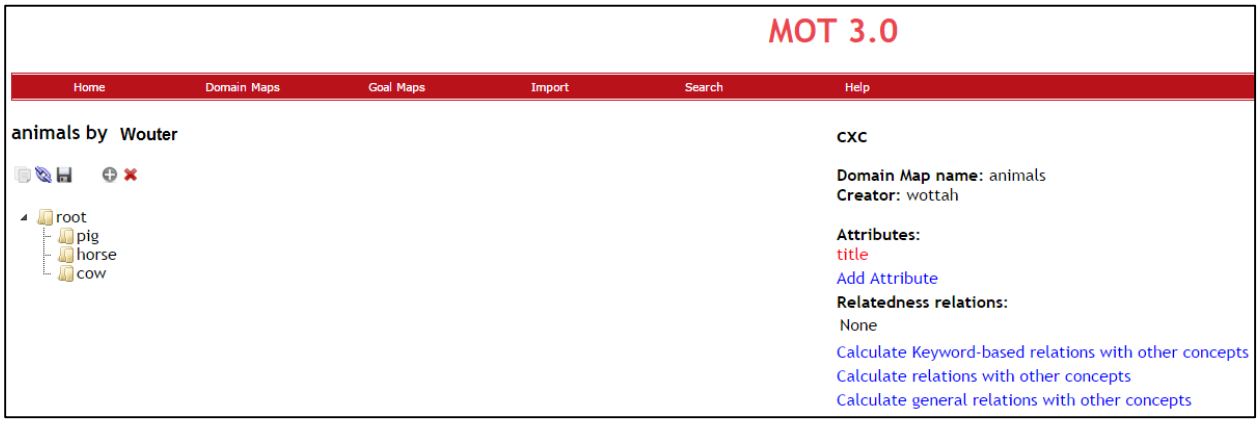

Figure 2.11: A snippet of the MOT 3.0 web interface

so that the author can create an entire adaptive application without having to deal with adaptation code of any kind.

The use of multiple tools and the lack of templating does make authoring in MOT more complicated. Even though PEAL provides support when creating the adaptation model, all standard behavior still has to be created separately for every concept. The use of different tools also creates a problem which can be seen in GAT as well: When an author wants to change the domain whilst creating the adaptation model, he then has to move back and forth between tools to achieve this.
A big advantage MOT has is platform independency. As ALAT can only be used to create GALE applications, MOT has been proven to be able to author for multiple adaptive hypermedia engines (either by compatibility or conversion) such as WHURLE, ADE and AHA! [CrSm05].

2.2.4 AMAS

Another more recent authoring environment is AMAS [HaCo11]. This is an adaptive educational hypermedia project in which ease of authoring and usability by non-experts are the main focus. It is designed to be usable by teachers. This is done by implementing reusable assets in terms of both content as well as adaptation strategies. Integrated group adaptation is implemented to help groups of students that are struggling with a particular topic. This kind of adaptation, while limiting the level of concept customization, greatly decreases the complexity of authoring an application. These features, which enhance usability and ease of authoring, are not present in ALAT. This partially has to do with the target audience selected for ALAT as described in *chapter 4*. However some of these supporting features are worth considering as future expansions and will be discussed in *chapter 7*.

Special attention in AMAS is paid to User experience (UX) as well. Gaffney, Conlan and Wade [GaCo14] claim that "*Inadequate UX design of AEH authoring tools may be a key factor impeding their widespread commercial and academic uptake*". This was measured by evaluating the style, color, look and feel and familiarity through questionnaires. Even though it might be difficult to measure and evaluate this quantitatively (apart from querying user opinions through surveys) it is clear that an effort has been made to make AMAS as visually appealing as possible (*figure 2.12*).

When comparing AMAS to ALAT, it is clear that there is a difference in target audience. But the uptake of ALAT by academic users could be an important project result. Even though UX has not been leading in the design of ALAT, the involvement of an HTI expert, as mentioned in *chapter 3*, will most likely have had a positive impact on the user experience. The following observations can be made with regards to the UX evaluation factors considered in [GaCo14]:

Figure 2.12: A screenshot of the AMAS interface. Taken from [GaCo14]

Style: *AMAS's style has been evaluated by asking the survey participants whether they liked the style or not. As the paper does not really give a clear definition as to what 'style' implies or how it is measured, it is difficult to form any relevant opinions regarding style in ALAT.*

Color: *The color scheme of ALAT has been kept calm and clean. Only a few colors are used to draw the interface (blue, white, black) in order to keep the interface uncluttered and minimalist. When comparing this to AMAS we can conclude that, though a different set of colors has been used, both interfaces use few colors. Bright colors and large contrasts have been avoided in both designs. Unfortunately the survey does not motivate its color scheme beyond the basic opinion of the participants.*

Look & Feel: *The survey on AMAS resulted in the following keywords to describe AMAS: 'orderly, 'calm', 'minimalist', 'functional' and 'focused'. Even though these keywords apply to ALAT as well at first glance, no surveys have been conducted to confirm this statement.*

Familiarity: *AMAS is evaluated in terms of familiarity by asking survey participants whether the interface reminded them of any other similar tools. The results are again hard to quantify and do not lead to more than a few statements and opinions. ALAT has been created using the Bootstrap[6] framework in order to create familiarity. Bootstrap is a popular HTML, CSS and JS framework. It is used to create a uniform and familiar style throughout any web application.*

No elaborate research has been done in order to create a UX which is fitted to the targeted audience. There has however been attention to the interface in order to make it easy to get started through familiar controls and to provide it with a clean and calm interface.

2.2.5 Discussion

This study has covered several current adaptive hypermedia authoring tools. What has become abundantly clear is that there are two major reason for the big differences between these authoring tools: Tool specialization and target audience. ACTSim and AMAS vary greatly in tool specialization with regards to ALAT. This also holds with regards to the target audience. MOT also differs from ALAT in that regard as it is created to be used by multiple authors. The tool in this comparison with similar specialization and target audience would be WOTAN. Here the graph interface and lack of separation of concerns are pitfalls which have been avoided as much as possible in ALAT.

Some of the solutions to difficult aspects of authoring could prove to be useful for ALAT. Additional support features implemented due to differences in tool specialization could also prove useful. A different design philosophy results in a different approach to problem solving. This difference in perspective is refreshing and could possibly benefit ALAT in future expansions.

[6] http://getbootstrap.com

Chapter 3

ALAT

The new authoring environment for GALE is called ALAT, an abbreviation for "*Adaptive Learning Authoring Tool*". Coincidentally it is also Indonesian for "tool". It is an entirely new authoring environment for GALE in which users can create domain and adaptation models with ease. This chapter describes ALAT and the important design decisions taken while creating it. It will also include a description of all important features and main workflow aspects. A reference to the previous authoring environments as described in *chapter 2* will be given where appropriate to explain to what extent features from the previous authoring tools have been recreated or redesigned.

3.1 The ALAT authoring interface

This section will focus on the design decisions of the ALAT interface. *Appendix A* will provide a more detailed description of the user interface layout and menus.
The goal of the ALAT authoring interface is to be easy and straight-forward to use. All operations available in ALAT are easy to perform without having to navigate through multiple elaborate menus and interfaces. The interface is also aimed to be user-friendly towards both novice adaptive hypermedia authors as well as more experienced authors. The design of the interface has been made in collaboration with the members of "De Roode Kikker" as well as a HTI (Human Technology Interaction) expert[7]. The resulting interface design (*Appendix B*) has been the foundation of the main ALAT screens.

Unlike both GAT and the AHA! graph author, there are no graph-like domain or adaptation model representations in ALAT. Graph interfaces tend to get cluttered quickly and make it difficult to keep track of the domain and adaptation model. Therefore the choice has been made to show all information either in lists or in a hierarchical structure. By navigating these hierarchies and filtering these lists it is easy to display the desired concept information without interface cluttering.

ALAT has a main concept hierarchy which is used to navigate through domain concepts in the user interface. This hierarchy is a tree-like structure which connects concepts through parent-child relations. These relations will also be defined in the resulting GALE application in order to allow further use of this structure. Using a main hierarchy is very different from the domain construction in GAT, which did not use a default relation and displayed all non-pedagogical relations together in a single canvas. ALAT takes the AHA! graph author approach which also used a tree-like project hierarchy to enforce a maintainable project structure. It is possible to add more (pedagogical and) non-pedagogical relations between concepts, but this functionality is moved to the concept settings screen so that the main tree structure is maintained.

[7] Yuexu Chen. She was active as an interaction designer at TU/e during the development of the ALAT project.

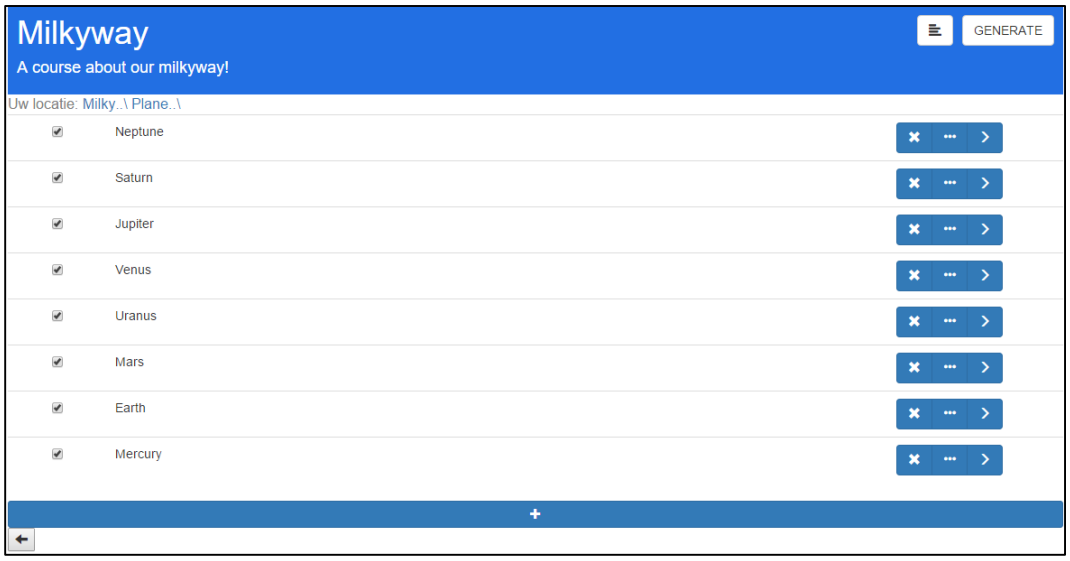

Figure 3.1: The step by step screen

All authoring functionalities in ALAT are divided amongst two main screens in which the domain and adaptation model can be constructed. There is a main screen which shows the current domain hierarchy and a concept settings screen which shows all concept details in a single overview.
The screen displaying the domain hierarchy has two view modes, which can easily be switched between with the click of a button. The first view (*figure 3.1*) provides a step-by-step domain navigation, reducing the number of concepts on the screen at any given time. The other view (*figure 3.2*), shows the domain hierarchy as a collapsible list, much like the AHA! graph author did. For novice users, the step-by-step view provides an easy way to navigate the domain hierarchy without causing an overload of information by displaying too many concepts on the screen at once. The list view is suited for more advanced users as the domain hierarchy is easy and fast to navigate and edit.

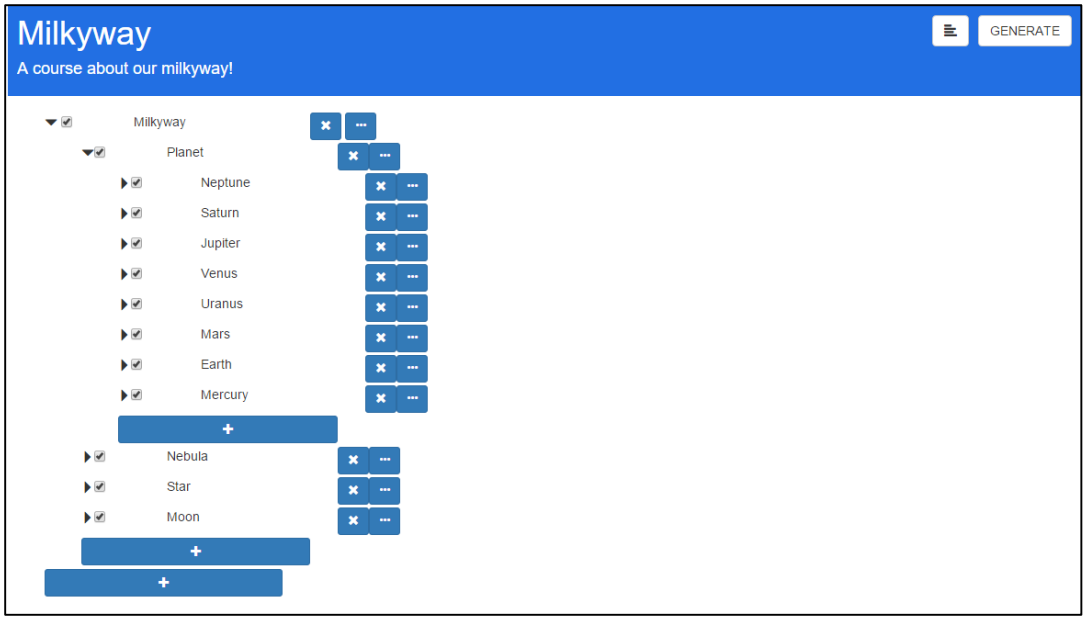

Figure 3.2: The overview screen

The second screen (*figure 3.4*), as mentioned before, shows all concept information in a single view. This screen can be easily opened and closed with a single click, minimizing the amount of menu navigating and context switching. It deals with adding, editing and removing concept properties, attributes, adaptation rules and non-pedagogical relations. These can all be managed using clean, straight-forward controls.
All concept attributes can be easily managed using the corresponding section in the settings window. Adding, removing and editing attributes is a quick and easy process which does not require further menu navigation.

The "*relations and expressions"* section of the screen is used to manage all pedagogical and non-pedagogical rules and relations. The user can choose to inspect and add a type of rule by using a drop down menu.
Tooltips are provided to inform the users about the function of adaptation rules and non-pedagogical relations. This section also enables the creation of new non-pedagogical relations. This is an easy process which only involves entering a name and tooltip as no GAM (GALE Adaptation Model) code is required to apply these relations. Once such a rule is created it is available in the filter list of any concept, so that it can easily be reused when needed.

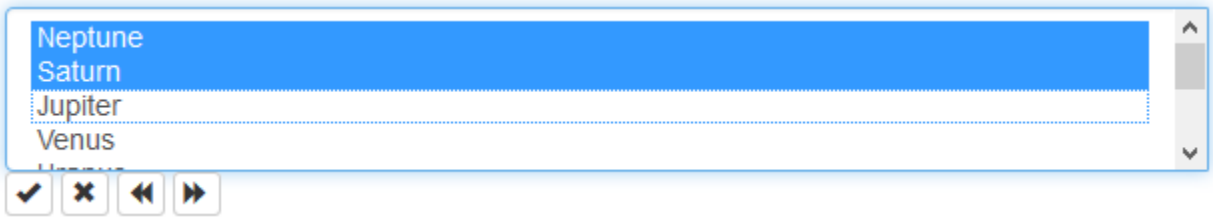

Figure 3.3: The rule target selection controls

A set of controls has been created in order to easily select targets for any given pedagogical or non-pedagogical relation (*figure 3.3)*. These controls can be used to walk through the project

hierarchy and select the desired rule targets. This solves the problems of huge lists of concepts, which was a big problem in GAT. A strong feature of GAT was the ability to apply adaptation rules on multiple concepts at once with the use of the socket interface as described in section 2. While it is no longer possible to select multiple sources for a single rule, there is the possibility to select multiple targets for any given pedagogical or non-pedagogical relation by means of multi-selection.

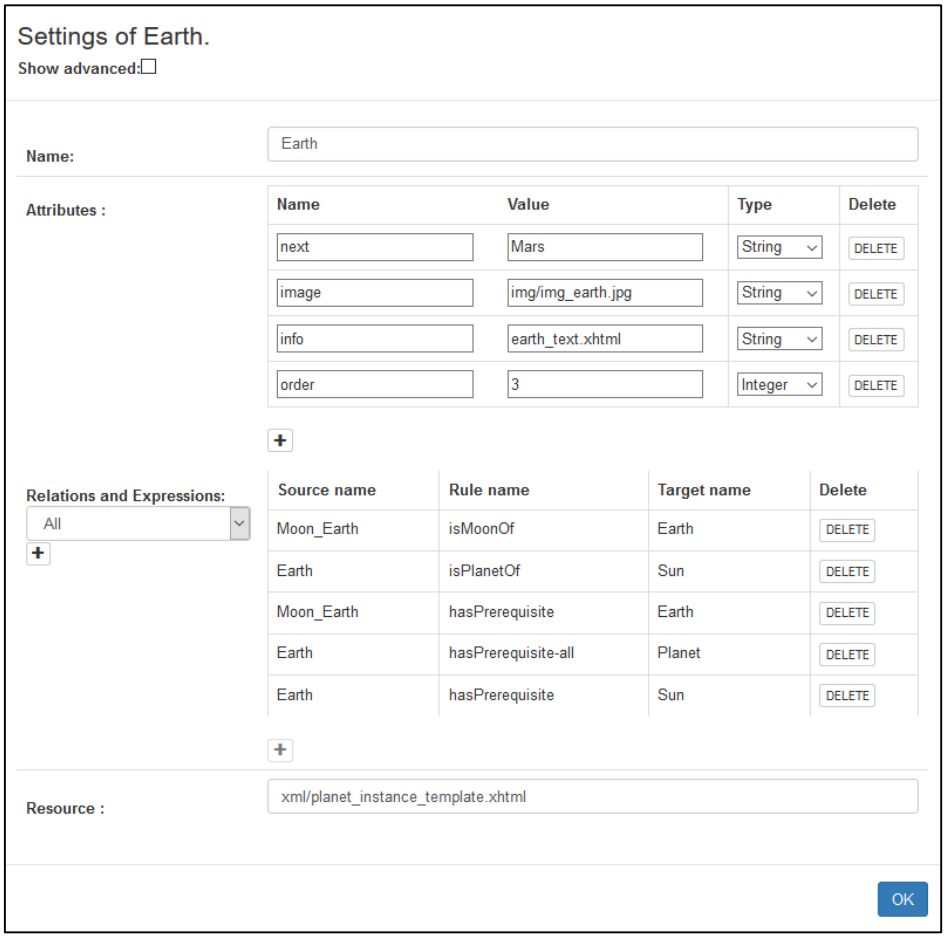

Figure 3.4: The settings screen

The user does not have to deal with any GAM code anywhere during the entire authoring process. As will be described in the next section, all concepts are created based on a templated blueprint. This means a selection of attributes and adaptation rules is applied when the concept is created. All data that originates from templates cannot be deleted from the concept. This is done to protect users from “breaking” the template. On top of that ALAT hides templated rules and attributes the user cannot edit. This prevents an overflow of information and confusion amongst novice users. More advanced users are able to display this hidden information by means of an “advanced mode” which can be toggled on and off.

3.2 Templating

One of the biggest improvements ALAT provides when compared to its predecessors is the ability to move the complicated scripting and technical aspects away from the user even more than the graph author and GAT. By extensively templating, a novice users with basic understanding of adaptive hypermedia will be able to author a course.
Templating to this extent is great for user-friendliness and workflow efficiency. It prevents having to apply standard behavior repeatedly and allows a set of attributes and concept adaptation rules to be set without confusing the author with its details.
An engineer with more advanced skills in GALE will be able to set up these templates and can edit and extend them when needed. The novice user will then be able to use these additions without having to deal with any GAM code.

The templates the tool uses cover a wide range of functionalities in order to support as much extendibility as possible. The templated data is divided in two JSON files; a concept blueprints file and pedagogical relation definitions file.

3.2.1 Format and structure

An appropriate storage format is necessary to store the templated information. This format must be able to efficiently store all required data. It must also be easy to parse by the authoring tool and easy to read for humans. Lastly it is important to choose an existing well-known format. This is preferable over a custom format as existing formats are more likely to be familiar to new users and are extensively documented. The main candidate formats have been XML and JSON, both of which are W3C[8] standards. XML stands for eXtensible Markup Language and is a language designed to be read by both humans and machines. It is structured to store and transport data.

JSON stands for JavaScript Object Notation. It is a format built for storing and exchanging data and is described as an easier to use alternative to XML.
JSON has been chosen over XML as it is easy to read and write. Its integration in rich web applications is intuitive and parses faster on servers than XML [NuAL09]. It is structured by declaring objects and arrays which can be read by ALAT without parsing.

3.2.2 In-application templating versus file editing

As described in the previous section, all templates are defined in JSON and are divided amongst two template files. The design decision has been made to craft these templates by hand as opposed to providing in-application support. This approach has been taken based on the evaluation of the AHA! graph author and GAT as described in chapter 2. GAT has a user interface for creating new adaptation rules, which is practical when a skilled author/engineer wants to add a new, non-existing rule on the fly. But this user interface approach is not practical when creating concept blueprints, or when creating an adaptation rule template setup. Elementary actions available in text editors, such as copying and pasting or searching and

[8] http://www.w3.org/

replacing are not possible. These operations would be very useful when performing operations such as creating variants of existing rules for example. It is simply faster and easier to just edit the properties file instead of having a graphical user interface. That is why the approach of the AHA! graph author has been taken. The result of this is that all templating has been moved outside of the application. The creation and maintenance of templates is efficient and fast because of the use of JSON, which is easy to understand and edit.

3.2.3 Blueprints

The concept blueprints are a set of basic concept types which are selected as a basic structure for new concepts. These types contain a set of concept attributes and adaptation rules. The blueprints file also contains a set of standard attributes which is applied to all concept blueprints. This is an extension to the extent of templating in the AHA! graph author. The reason for the standard attributes is that GALE makes use of adaptive views, such as the “static-tree-view” or the “next-view” [SmBr11] which use these standard variables for their adaptive behavior. These views are usually part of a layout in which concepts are presented.

By having these attributes apply to all concept blueprints it is much easier to use multiple concept blueprints within the same view. The remainder of this section will cover the template format and content.

An example blueprint template is shown in *figure 3.5*. Two main objects are declared in this template file; “*defaultAttributes*” and “*conceptTypes*”. These contain the default attributes and the concept blueprints respectively. The attributes consist out of an attribute name, data type (Boolean, Integer, Double or String) and a default value. Attributes with the Double or Integer type contain an extra parameter which is used to resolve multiple adaptation rules applied to this attribute. The current version of ALAT supports “SUM” and “AVG” to either sum up the outcome of all expressions or take the average of the resulting adaptation expression values. The source code is set up for future expansions such as using a minimum (MIN) or maximum (MAX) value.

```
{
   "defaultAttributes":[
      {
         "name":"suitability",
         "type":"Boolean",
         "value":"true"
      },
      {
         "name":"knowledge",
         "type":"Double",
         "value":"0",
         "operator":"AVG"
      }
   ],
   "conceptTypes":[
      {
         "name":"text-topic",
         "default_attributes":[
            {
               "name":"info",
               "type":"String",
               "value":"This is some basic concept information text!"
            }
         ],
         "default_rules":[
            "own knowledge update",
            "visited",
            "knowledge_update"
         ]
      }
   ]
}
```

Figure 3.5: An example blueprints template

The object containing the concept types in this case contains the definition of a single concept blueprint. It consists out of a template name and the sets of concept attributes and adaptation rules as mentioned before. The concept attributes are defined much like the default. The main difference is that these attributes do not support multiple rules to be applied to them. The reason for this lies in the way adaptation rules are applied. The attributes targeted by adaptation rules do not need to be declared separately in the concept blueprint. Even though this limits the possibility of applying multiple rules to a non-default attribute, it prevents having to declare every attribute twice. This would have made creating and editing template data confusing and complicated.

These concept attributes are meant to be used in page templates[9] or to be used as parameters for adaptation rules, e.g. a minimum amount of right answers required to pass a test.

As can be seen in the template example, the adaptation rules section of the concept blueprint does not contain actual adaptation rule definitions. All rule and relation definitions are stored in a separate template. This makes all adaptation rules reusable and prevents redundant rule declarations.

Having blueprints takes care of standard concept structures without having to set these manually and without having to know their details. On top of that it prevents redundant work in setting up this structure multiple times for each new concept. In other words; the concept blueprints benefit both the entry barrier as well as the authoring workflow efficiency. The standard attributes can be edited in order to take in account possible future updates to GALE views. These new views and layouts might use a different set of default properties for their adaptive behavior. The concept blueprints template ensures the tool remains usable and can be brought up to date as new content is created for GALE.

3.2.4 Relation Definitions

The pedagogical relation definitions templates define all rules and relations that exist within ALAT. The purpose of the authoring tool is to have the user author adaptive hypermedia content without having to deal with any code or definition files. It is also important that rule definitions can be edited, extended and created. For this reason all rules and relations are defined within a template. Though moving all code and definitions away from the user limits the versatility and variety of possible rules within ALAT, it provides users with rules which are easy to comprehend and which are even easier to apply.

An example of how a rules and relations template is structured is provided in *figure 3.6*.
Three types of rules and relations are defined in this example template. These can either be applied manually by adding them to concepts in the authoring interface, or they can be applied automatically by being included in a concept blueprint template. The distinction between different types of rules and relations can be made based on their application.

[9] The use of page templates will not further be covered in this Thesis. [SmBr11] provides an in-depth description.

```
{
   "def_att_rules":[
      {
         "name":"hasPrerequisite",
         "type":"binary",
         "target":"suitability",
         "tooltip":"Target concept must be learned before source is recommended.",
         "code":"${%target%#knowledge} > 0.8",
         "operator":"and"
      }
   ],
   "persistent_att_rules":[
      {
         "name":"visited",
         "type":"unary",
         "properties":[
            {
               "name":"visited",
               "type":"Integer",
               "defval":""
            }
         ],
         "tooltip":"stores number of concept visits in ‘visited’",
         "code":"#[visited]:Integer event +`if (${#suitability}) { ${#visited}++;}`"
      }
   ],
   "def_relations":[
      {
         "name":"rotatesAround",
         "tooltip":"source concept rotates around the target object."
      }
   ]
}
```

Figure 3.6: An example pedagogical relation definitions template file

First there are rules that apply to the default attributes defined in the blueprint template. These rules are different from others because multiple conditions and rules can be applied to a single default attribute. They usually consist out of a name, type, target, tooltip and the rule GAM code and are defined within the “*def_att_rules*” object. The rule type can either be “*unary*” or “*binary*”. ALAT uses this parameter to know whether a rule targets another concept or not. This is necessary because ALAT does not interpret the GAM expressions set in the template. By having this type distinction ALAT will have the user select a target concept for binary rules. This will be applied by replacing *“%target%”* in the GAM expression by the actual target concept name.

The “*target*” attribute refers to the name of the default attribute this rule is targeting.
Rules targeting Boolean default attributes get another parameter called “*operator*”. This is used

to indicate whether this rule should be declared in an "*AND*" or an "*OR*" clause. This can be used for special kinds of prerequisites, for example. Multiple "*OR*" and "*AND*" clauses can be applied to the same standard attribute.

Example: We take three rules called "*OrRule1*", "*OrRule2*" and "*AndRule1*" targeting Boolean attribute "*suitability"*. "*AndRule1*" has the "AND" operator applied to it whereas "*OrRule1"* and "*OrRule2"* have the "OR" rule apply to them. When applying these three rules the resulting GAM code would be: "*#suitability:Boolean = `true && AndRule1 && (false | OrRule1 | OrRule2) `*". This combines all three rules into a single GAM-expression. Now "*suitability"* will have a "*true"* value if "*AndRule1"* values true and at least one of the two "OR" rules is true.

The second type or rules within the template consists of attributes which are defined by a single GAM expression. They are contained within the "*persistent_att_rules"* object. These rules can thus not use the "operator" parameter as only a single rule can be applied to the attribute targeted. Instead these rules do have a set of attributes which are defined in the rule "properties" array. These attributes should include the target attribute the rule expresses as well as all other attributes which are used as parameters in this rule. The attributes declared here consist out of a name, type and default value. As ALAT does not interpret the GAM code, the default value of the targeted attribute should remain blank. This tells ALAT that the value of this attribute is defined by the GAM code.

Lastly there are the non-pedagogical relations. These are defined within the "*def_relations"* object. These relations do not have any particular adaptation behavior attached to them, but are labeled relations between two concepts (as described in *chapter 1.3*). They consist out of a name and tooltip. No other parameters are required as no attributes can be targeted by these rules and they are binary by default.

The implementation of this template in the authoring environment provides easy and clear rules which can be added by the click of a button without any GAM code knowledge required. All rules and relations are supplied with tooltips in order to prevent misinterpretation and to make them easier to understand.

3.3 The ALAT system architecture

This section deals with the ALAT system architecture and technical aspects. ALAT is a browser-based web application with HTML and Javascript running in a browser screen. This is a lightweight, well-documented and platform-independent solution which is up to current web standards. The AngularJS framework has been chosen in order to develop dynamic and extensible Javascript code. AngularJS is a framework which provides the tools necessary to create dynamic views in which rapidly changing data can be displayed with ease.

AngularJS also provides the tools for a layered object orientated approach which aids in the creation of maintainable extensible code. The ALAT client side application layer consists out of a 3-tier model (*figure 3.7*): A controller layer in order to manage the views displayed in the corresponding HTML page, a service layer for data storage and manipulation and a data layer

which communicates with the server-side of the ALAT system. A more elaborate version of this architecture can be found in *Appendix C*.

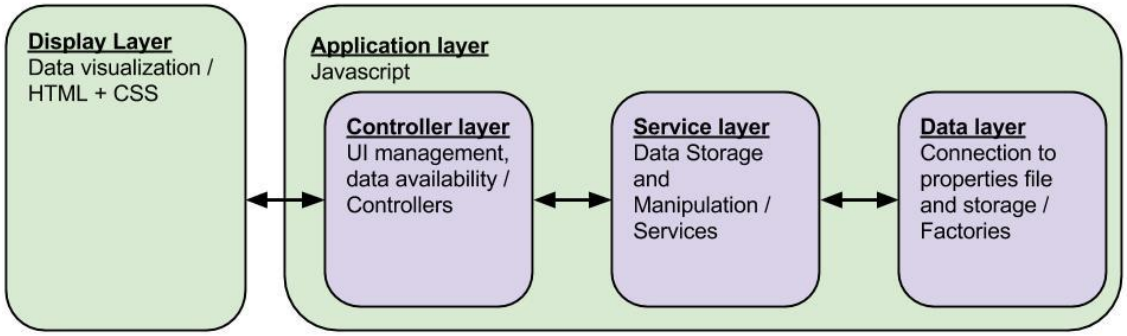

Figure 3.7: The ALAT client-side Application Architecture

The server-side of ALAT (*figure 3.8*) uses PHP to manage database and storage operations which are used to manage user account and project information as well as the actual project data. The user account and project information is stored in a MySQL database. The project data is stored in JSON files.

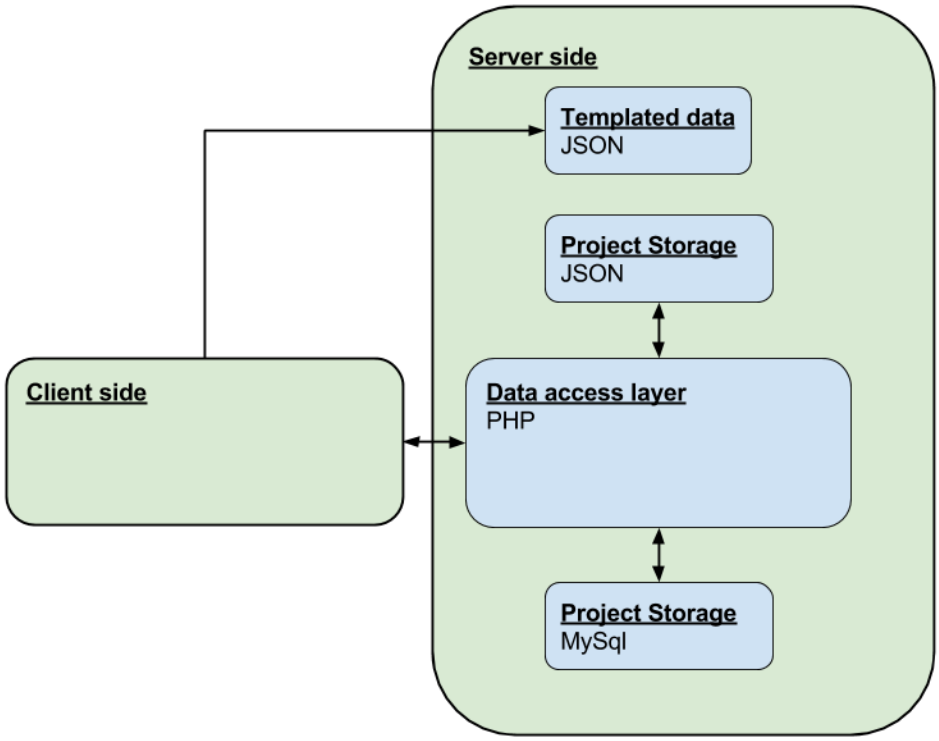

Figure 3.8: The ALAT Back-end Architecture

3.3 Generating GALE applications

After a GALE application has been designed using the graphical user interface it is ready to be generated and deployed. In ALAT this is done by using a “generate” button. This single-click operation starts both the generation of the GALE application by generating a resulting “concepts.gam” file as well as the deployment of this application on the GALE server. This makes the creation of GALE applications in ALAT very easy and helps the user deploy his application without requiring any knowledge of the GALE server workings or the deployment procedure.

3.3.1 Generating a GAM file

Before the GALE application can be deployed, its GAM file first has to be generated. This section will deal with all relevant steps ALAT takes when generating this file. Unfortunately, this process is more complicated than simply printing out the data structure created in ALAT and a series of steps has to be taken in order to create the right GAM format containing the appropriate information.

The first thing that needs to be taken care of is printing the concept templates. The list of concept blueprints is first filtered by checking which of them are used within the ALAT project. There is no need to print concepts that are not used and doing so would result in unused GAM code which complicates the resulting GAM file and could also be considered to be ‘unsanitary’. After this selection is made, the used concept blueprints are translated to GAM code. When this is completed the system outputs all selected project concepts to GAM code as well. The ALAT interface has the option to include or exclude parts of a concept tree. *Figure 3.9* shows a project in which only the concept hierarchy of “*Plants”* and “*Animals”* would be generated and deployed. This enables applications with a select part of the designed ALAT project to be generated and deployed.

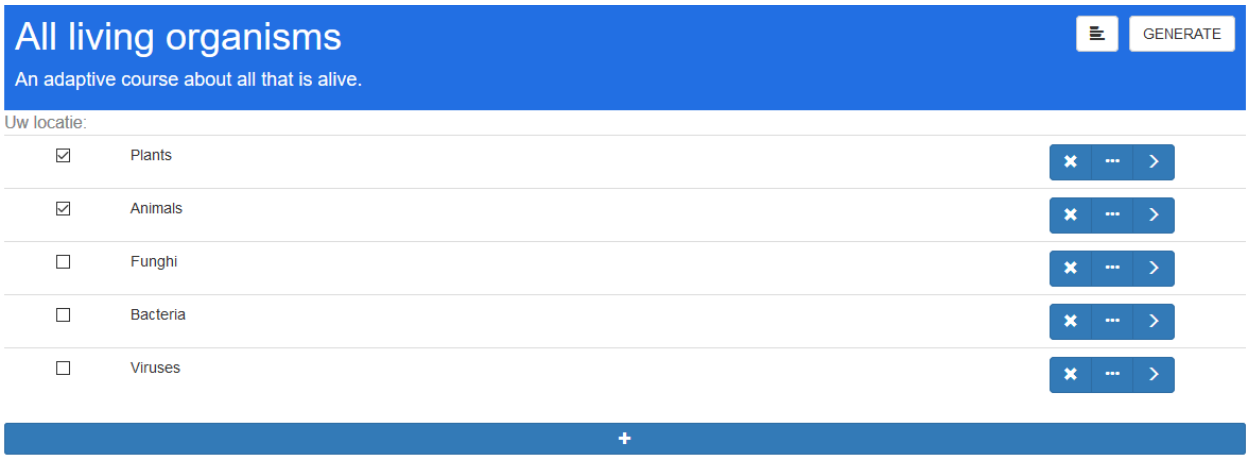

Figure 3.9: An ALAT project set to be only partially deployed

The concept blueprint behavior is applied to these concepts by means of inheritance. In GAM this inheritance is created by applying “*->(extends)blueprintName”* to the concept code, where “*blueprintName”* is the name of the concept blueprint selected for the concept. This is a relation which enables the inheritance of

concept behavior. The project structure as created in the ALAT user interface is enforced by applying relations with a “parent” label to all concepts.

It is pretty straight-forward to translate attributes, parameters and simple attribute expressions defined within concept blueprints and actual concepts to the GAM syntax. Printing the standard attributes however, is more complicated.

Standard attributes, as mentioned before, are attributes that can have multiple rules and relations applied to them. A list of all these applied rules has to be made before their GAM code can be created. When this list is constructed, all operators as described in section 3.2.3 and 3.2.4 can be applied and the GAM output can be created.
Creating GAM code for concepts other than the used blueprints takes an extra step. Attribute relations and rules inherited from the concept blueprints can be overwritten by declaring the attribute again in the current concept. Therefore when a concept extends the rules on standard attributes as declared in its blueprint, it is required for all inherited rules to be applied to the attribute once more. This has to be done in order to maintain the blueprinted behavior.

When the resulting GAM code has been created, it is sent to the data access layer on the server.

3.3.2 Deploying GALE applications to the server

After the ALAT project has been translated to a GAM file it is passed through to the data access layer by means of AJAX requests. This layer creates a unique folder within the GALE application directory using the project and author name. The data layer writes the GAM code into a file named “concepts.gam” which is then deployed to this directory. If the directory already exists, the existing “concepts.gam” is replaced with the new version.

Also stored within the data layer are a reference to a default layout, appropriately named “layout.xhtml” as well as a placeholder file named “placeholder.xhtml” which is used for concepts without any declared content resource. After the “concepts.gam” file is created a copy of the layout and the placeholder are placed in this folder as well. This results in a deployed GALE application in which it is easy for experts to extend and edit both the GALE application itself by editing the GAM code, as well as the default layout and placeholder by editing their appropriate files. All these files are conveniently located within a single directory.

Chapter 4

Target audience and applications

In this chapter the target audience of ALAT will be described. A description of the target audience also allows for a discussion on the possible applications of ALAT. With these applications in mind a case is made on the possibility of future expansions of ALAT and what is needed to reach new audiences.

4.1 The ALAT target audience and applications

ALAT has been created in order to provide a new authoring environment for GALE. At this time GALE and GAT are mainly used in courses and projects at TU/e. The first possible application of ALAT is to be the new recommended authoring environment in the adaptive hypermedia courses at TU/e. This means ALAT will be used by students with a technical background who are not yet experienced in authoring adaptive applications. ALAT provides the tools necessary to ease these students into authoring these applications while they learn the principles of adaptive hypermedia. This means that the main target audience of ALAT at this time consists of both students learning to author for GALE as well as engineers and experts with more extensive knowledge on adaptive hypermedia.

The ALAT authoring environment as it is now is created to be used by authors with at least basic understanding of adaptive hypermedia with the support of a GALE expert. An expert is to set up the templates needed to create adaptive applications. It might be difficult for users to start using ALAT without the expert in the background, because extending templates might be necessary to create the desired adaptation rules or concept blueprints. A solution to this would be to create an all-round and broad set of templates covering all common adaptation techniques used in GALE.
An effort has been made to make ALAT a usable authoring environment for more adept GALE users as well. Experienced GALE users can install ALAT on their own server and can create their own templates and default layout files. Even though experts might prefer writing their own GALE code, deployed ALAT applications could serve as a basis for further development. This is something which is not possible in GAT, because it deploys applications in the IMS VDEX[10] format [Sm11], which is not easy for users to read and edit.

4.2 Possible future applications

As described earlier, ALAT will mainly be used by students and GALE experts at TU/e in the foreseeable future. This section will discuss what might hold ALAT back from different audiences and what solutions could possibly solve this problem. Because of the collaboration with “De Roode Kikker” throughout this project, adaptive educational hypermedia (AEH) is regarded to be the most prominent future expansion at this time.

[10] http://www.imsglobal.org/vdex/

Discussions on ALAT and AEH authoring with members of “De Roode Kikker” have resulted in an interesting analysis on authoring adaptive software in the field of education. AEH is an important possible future application of ALAT. This would result in educational hypermedia experts and possibly teachers as an additional target audience.

It is abundantly clear that academic users and educational hypermedia experts have different desires and needs when it comes to authoring adaptive applications. As opposed to academic users, the educational expert has no knowledge about the workings of GALE at all. Nor is this expert likely to be interested in this. The interface should match this by displaying information in a way that corresponds to the users’ way of thinking, rather than matching a resulting GALE domain model. This also holds for the terminology within the authoring environment. As these experts might use the authoring environment more regularly, a slight learning curve is acceptable. However, the more advanced features should not confuse new users trying to use the main authoring features. A more extensive “advanced mode” could help prevent this confusion.

For teachers trying to play around with adaptive hypermedia the tool should be simplified even more. These users will probably use the authoring tool sporadically and might forget some of the gained knowledge authoring in between uses. A version of ALAT geared toward use by teachers and non-technical users should probably rely on a few concept blueprints for adaptivity. A new user interface should be designed to make authoring a process which is more visually appealing and is adjusted to the way teachers think about creating courses.

So the main factors contributing to the usability of authoring tools such as ALAT for teachers or educational hypermedia experts are: user friendliness, an appropriate level of complexity and an interface which lets users construct courses in a way that matches their ideas on course construction. Academic users will possibly desire a more diverse set of adaptation rules and concept blueprints to experiment and learn with. Educational hypermedia experts and teachers will likely need a subset of rules and concept blueprints that apply to the field of application in particular. Especially for teachers, it is necessary to keep the number of rules to a minimum in order to prevent information overload and a confusing number of possibilities.

Chapter 5

Conclusion

Authoring is a difficult subject within the field of adaptive hypermedia. Numerous research projects have resulted in authoring tools in an attempt to make authoring adaptive applications easier. In the case of GALE this has resulted in a tool called GAT. This thesis presents ALAT, which is a successor to GAT.

ALAT is to be the new recommended authoring environment for GALE. Preceding projects have been analyzed in order to avoid pitfalls and mistakes made in the past and to find what is required to make ALAT a successful authoring tool. ALAT is a tool which is aimed to be as generic as possible whilst providing the user with the best possible support to author adaptive applications. This is done through templating, which has been applied to an unpreceded extent in order to improve usability. It also allows authors to author adaptivity without any required knowledge on adaptation code.

The target audience consists of students and engineers with at least a basic understanding of adaptive hypermedia. A result of this is that ALAT is more difficult to author with for non-academic users. This is due to the complexity and user experience which is a result of ALAT's extensive generic behavior. A version of ALAT with a limited scope, stripped of some more advanced features would lower this entry barrier and could make ALAT more suitable for non-experts and companies such as "De Roode Kikker".

ALAT contributes to the usability of GALE and brings a new player to the field of adaptive hypermedia authoring. This thesis provides an in-depth documentation and analysis of ALAT and its main features. A comparative study shows that there are numerous approaches to adaptive hypermedia authoring. The main contributing factors to these differences are target audience and genericity. ALAT explores authoring by academic users and innovates by combining a simple user interface with extensive templating. This results in a generic platform in which it is easy to author various kinds of adaptive hypermedia applications.

Chapter 6

Future Work

Future expansions and research on ALAT can be split into two categories: “Technical improvements and additions” and “research and development”. The first category involves improvements and additions which can be implemented in ALAT in order to enhance stability, authoring support and user experience. The second category consists of research topics which could then lead to further development and expansion of ALAT.

6.1 Improvements & Additions

A number of improvements can be made to ALAT without the need of further research. Most of these improvements and additions have not been made due constraints in time and manpower or project focus. The most prominent of these improvements is extensive testing. ALAT has been tested by creating example applications and performing all possible actions currently available. Yet no extensive test cases or unit tests have been set up in order to prove the correctness of ALAT’s functionalities and to find remaining bugs. In addition to this, the server-side of ALAT can be extended by adding more account security as well as public projects (accessible by multiple users). Lastly, ALAT’s style could be further improved with the help of user interface experts in order to create a more polished look and feel.

Tests should also be designed and performed in order to verify the models generated by ALAT. Correctness of the resulting GALE application is a key functionality and should always result in a correct GALE application (provided the templates do not contain any errors).

A number of applications exploiting all different functionalities of ALAT should be created to showcase its capabilities and to give an indication as to how a finished project in ALAT looks. These showcase applications might prove valuable to provide insight in adaptive hypermedia authoring for new users.

6.2 Research & Development

Further research could be conducted to determine a set of broad and general adaptation rules and concept blueprints which can serve as a foundation for new ALAT users. A closer examination of possible default Layout files for applications developed in ALAT could go in tandem with this research and development. More support for the selection of layout files or (sets of) concept blueprints can then be developed.

Extensive research in the field of learning styles in adaptive hypermedia has already been conducted at TU/e [St07] [StCr06]. This information can be used to perform new research projects in order invent new ways to author and apply learning styles in adaptive hypermedia. These new methods and techniques can be turned into a development project for ALAT. New modules can be developed in order to provide easy controls for authoring learning styles. This would especially benefit ALAT in the area of e-learning.

A field which is not as thoroughly explored for GALE or its predecessors is group adaptation. Group adaptation could greatly benefit adaptation possibilities in ALAT. It is used to provide adaptation based on a group defined in the user model. Providing support for group adaptation in ALAT would greatly improve the amount of adaptivity possibilities whilst keeping ALAT generic and easy to author with.

A version of ALAT dedicated to be used for AEH purposes is a future expansion which could prove especially useful in the cooperation between TU/e and “De Roode Kikker”. As discussed in *chapter 4*, educational hypermedia experts and teachers require a different authoring environment than academic users. Research could be conducted in order to find out what kind of project visualization, templates and user experience are required to create a version of ALAT suitable for these users.

Graphical representations of the Adaptation and Domain Model have been a difficult subject in multiple related projects such as WOTAN and GAT (*section 2*). ALAT in its current state uses two hierarchical list-like views to represent its domain and adaptation models. Future research can be conducted in model visualization techniques for ALAT in order to create graphical visualization options for domain and adaptation models in ALAT. Graphical visualization is a topic which is also of great interest of “De Roode Kikker” because a good visual implementation is much more appealing and is often easy to understand as opposed to a more textual format.

More topics for research and development might start appearing as ALAT is taken in use. Student suggestions might result in a wide variety of new conceivable support features to further increase the possibilities of adaptation techniques in ALAT.

Bibliography

[BrAl13]
De Bra, Paul, et al. "GRAPPLE: Learning management systems meet adaptive learning environments." Intelligent and adaptive educational-learning systems. Springer Berlin Heidelberg, 2013. 133-160.

[BrEl96]
Brusilovsky, Peter, Elmar Schwarz, and Gerhard Weber. "ELM-ART: An intelligent tutoring system on World Wide Web." Intelligent tutoring systems. Springer Berlin Heidelberg, 1996.

[BrLe93]
Brusilovsky, Peter, Leonid Pesin, and Mikhail Zyryanov. "Towards an adaptive hypermedia component for an intelligent learning environment." Human-computer interaction. Springer Berlin Heidelberg, 1993. 348-358.

[BrSc96]
Brusilovsky, Peter, Elmar Schwarz, and Gerhard Weber. "A tool for developing adaptive electronic textbooks on WWW." Web site: http://aace. virginia. edu/aace/conf/webnet/html/151/151. htm, 1996.

[BrSm06]
De Bra, Paul, David Smits, and Natalia Stash. "The design of AHA!."Proceedings of the seventeenth conference on Hypertext and hypermedia. ACM, 2006.

[BrSm12]
De Bra, Paul, and David Smits. "A fully generic approach for realizing the adaptive web." SOFSEM 2012: Theory and Practice of Computer Science. Springer Berlin Heidelberg, 2012. 64-76.

[CrMo03]
Cristea, Alexandra I., and Arnout De Mooij. "LAOS: Layered WWW AHS authoring model and their corresponding algebraic operators." WWW03 (The Twelfth International World Wide Web Conference), Alternate Track on Education, Budapest, Hungary. 2003.

[CrSm05]
Cristea, Alexandra, David Smits, and Paul de Bra. "Writing MOT, Reading AHA!." Converting between an authoring and a delivery system for adaptive educational hypermedia, University of Technology, Eindhoven (2005).

[CrSm09]
Cristea, Alexandra I., et al. "LAG 2.0: Refining a reusable Adaptation Language and Improving on its Authoring." Learning in the Synergy of Multiple Disciplines. Springer Berlin Heidelberg, 2009. 7-21.

[FoCr10]
Foss, Jonathan GK, and Alexandra I. Cristea. "The next generation Authoring Adaptive Hypermedia: Using and Evaluating the MOT3. 0 and PEAL tools."Proceedings of the 21st ACM conference on Hypertext and hypermedia. ACM, 2010.

[FrNe98]
Fröhlich, Peter, Wolfgang Nejdl, and Martin Wolpers. "KBS-Hyperbook-an Open Hyperbook System for Education." Proceedings of the ED-MEDIA World Conference on Educational Multimedia and Hypermedia. 1998.

[FrRo05]
Freire, Manuel, and Pilar Rodríguez. "Comparing graphs and trees for adaptive hypermedia authoring." A3EH: Third International Workshop on Authoring of Adaptive and Adaptable Educational Hypermedia, at 12th International Conference on Artificial Intelligence in Education AIED. 2005.

[GaCo14]
Gaffney, Conor, Owen Conlan, and Vincent Wade. "The AMAS authoring tool 2.0: a UX evaluation." Proceedings of the 25th ACM conference on Hypertext and social media. ACM, 2014.

[GaDa10]
Gaffney, Conor, Declan Dagger, and Vincent Wade. "Authoring and Delivering Personalised Simulations-an Innovative Approach to Adaptive eLearning for Soft Skills." J. UCS 16.19 (2010): 2780-2800.

[HaCo11]
Hampson, Cormac, Owen Conlan, and Vincent Wade. "Challenges in Locating Content and Services for Adaptive eLearning Courses." Advanced Learning Technologies (ICALT), 2011 11th IEEE International Conference on. IEEE, 2011.

[NuAL09]
Nurseitov, Nurzhan, et al. "Comparison of JSON and XML Data Interchange Formats: A Case Study." Caine 9 (2009): 157-162.

[QuNe02]
Qu, Changtao, and Wolfgang Nejdl. "Bridging O-telos and XML with XML schema: the authoring environment for KBS Adaptive Hyperbook." Information Technology: Coding and Computing, 2002. Proceedings. International Conference on. IEEE, 2002.

[Sm11]
Smits, D. David. “Towards a generic distributed adaptive hypermedia environment.” Technische Universiteit Eindhoven, 2012.

[SmBr11]
Smits, David, and Paul De Bra. "GALE: a highly extensible adaptive hypermedia engine." Proceedings of the 22nd ACM conference on Hypertext and hypermedia. ACM, 2011.

[St07]
Stash, Natalia. "Incorporating cognitive/learning styles in a general-purpose adaptive hypermedia system." Dissertation Abstracts International 68.04 (2007).

[StCr06]
Stash, Natalia, Alexandra Cristea, and Paul De Bra. "Adaptation to learning styles in e-learning: Approach evaluation." World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education. Vol. 2006. No. 1. 2006.

Appendixes

Appendix A

The ALAT User Interface

Figure A.1 and *figure A.2* show the login screen and the register screen. These are the first screens the user sees when starting ALAT.
After logging in the user will arrive at the project browser, which can be seen in *figure A.3*. Here the user can either select, create or delete projects. After a project is opened, the main ALAT interface will appear.

Figure A.4 and *figure A.5* show the controls for adding new concepts to the current hierarchy. *Figure A.6*, *figure A.7*, *figure A.8* and *figure A.9* show the details of the settings screen. The screenshots in *figure A.6* and *figure A.7* show the difference between adding a unary or binary rule. The rule tables in these figures are (nearly) empty as selecting a rule will filter the table, showing only the rules of the type which is currently selected.

Figure A.8 shows what adding a new attribute to the current concept looks like.

The advanced settings screen in *figure A.5* shows a number of rules and attributes for which a number of fields are disabled. This is done so that the templated data cannot be changed, as mentioned in *chapter 3*.

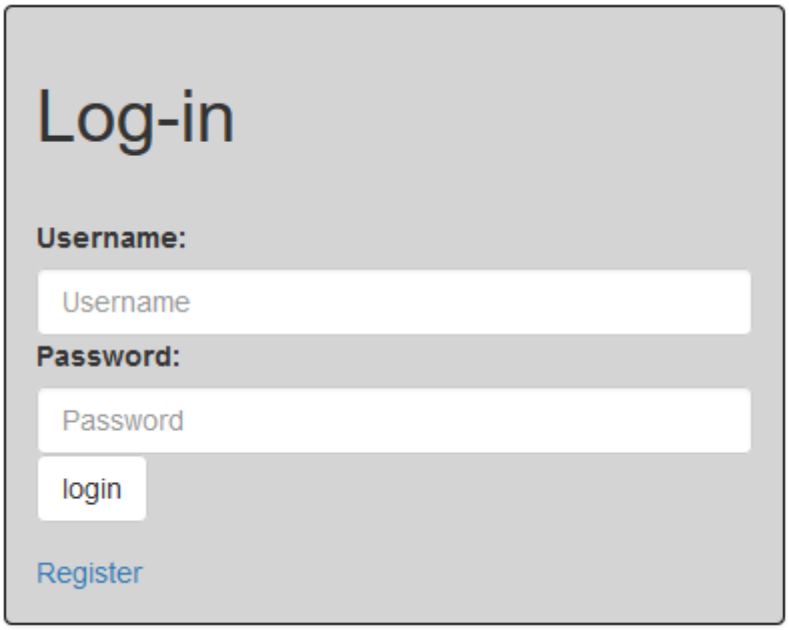

Figure A.3: The login screen

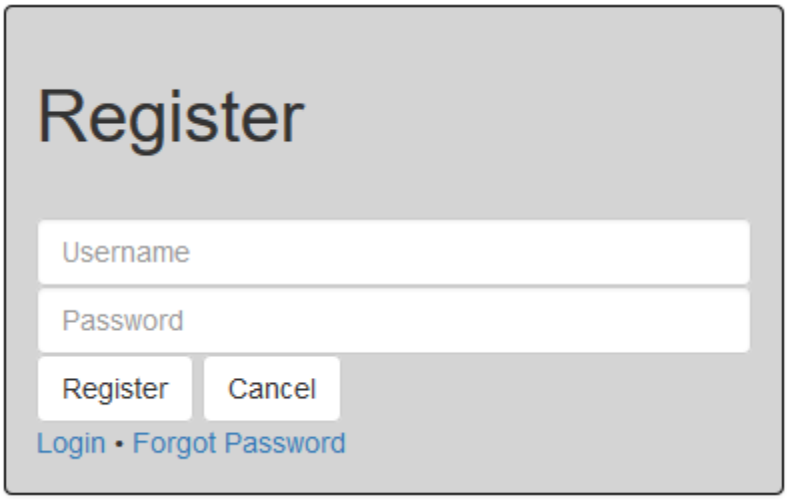

Figure A.1: The register screen

Figure A.2: The project selection screen

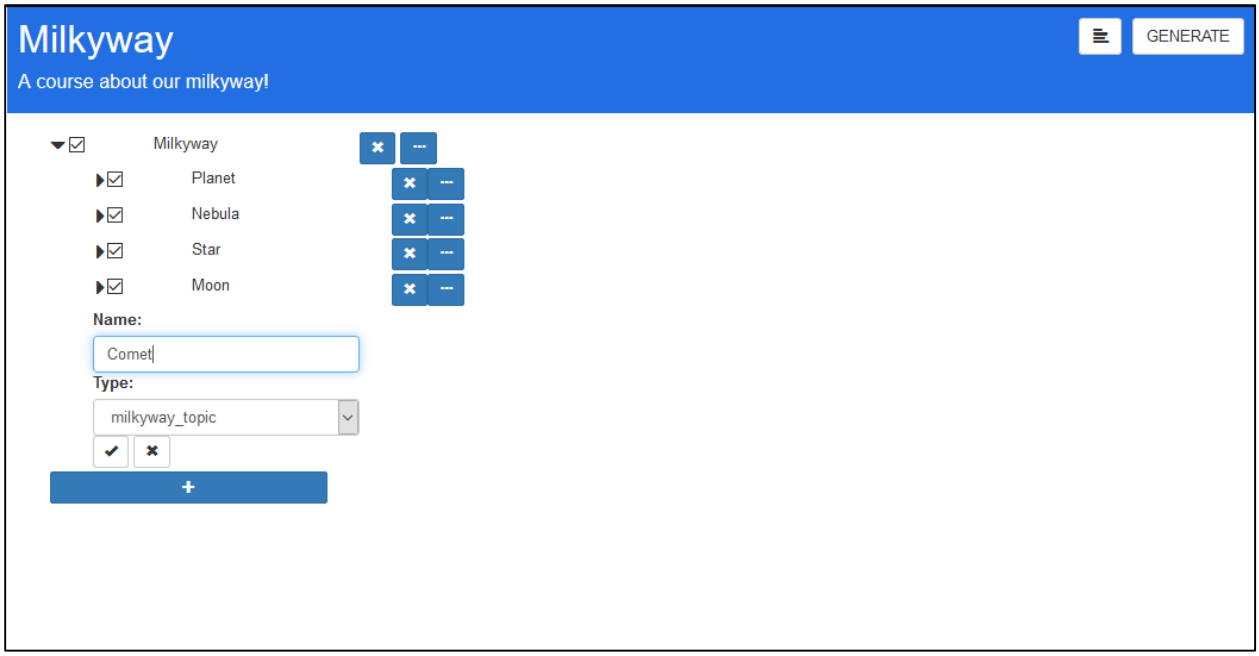

Figure A.4: Adding a concept in overview mode

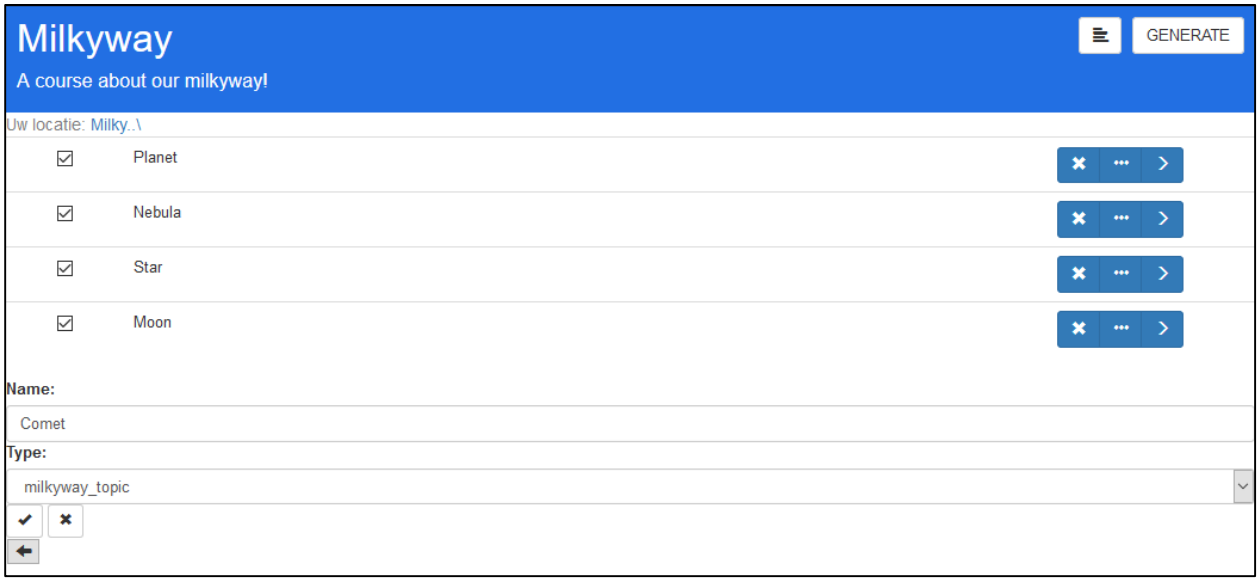

Figure A.5: Adding a concept in the step-by-step view

Figure A.7: Adding a unary rule

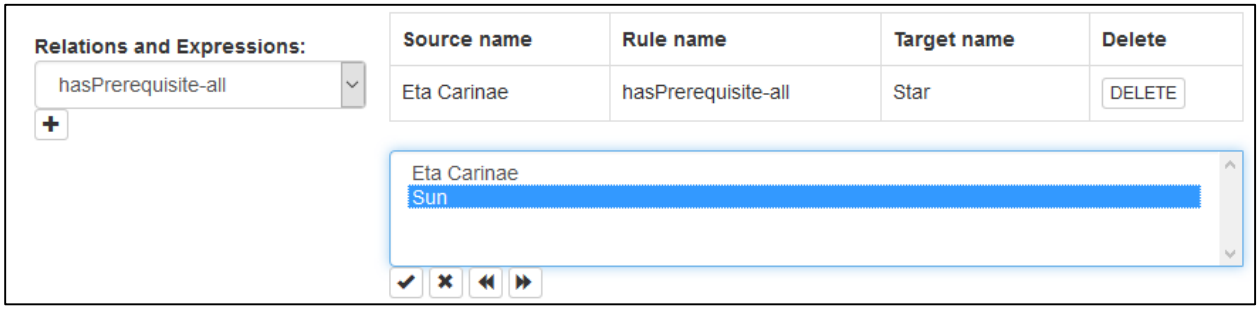

Figure A.8: Adding a binary rule

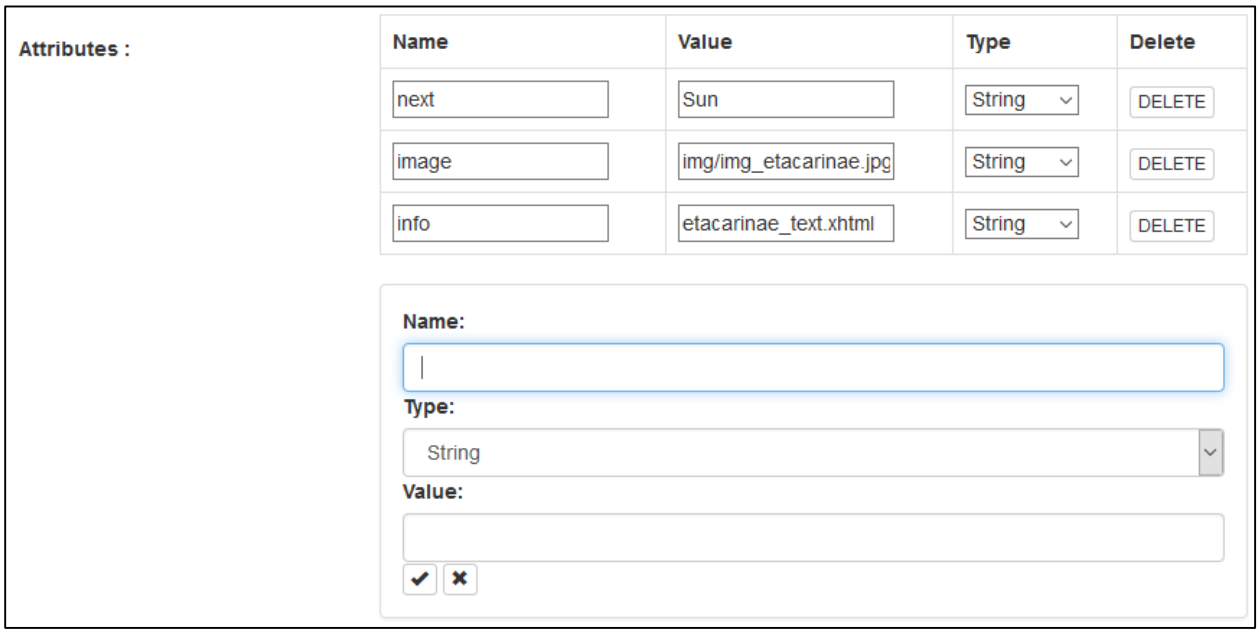

Figure A.6: Adding an attribute

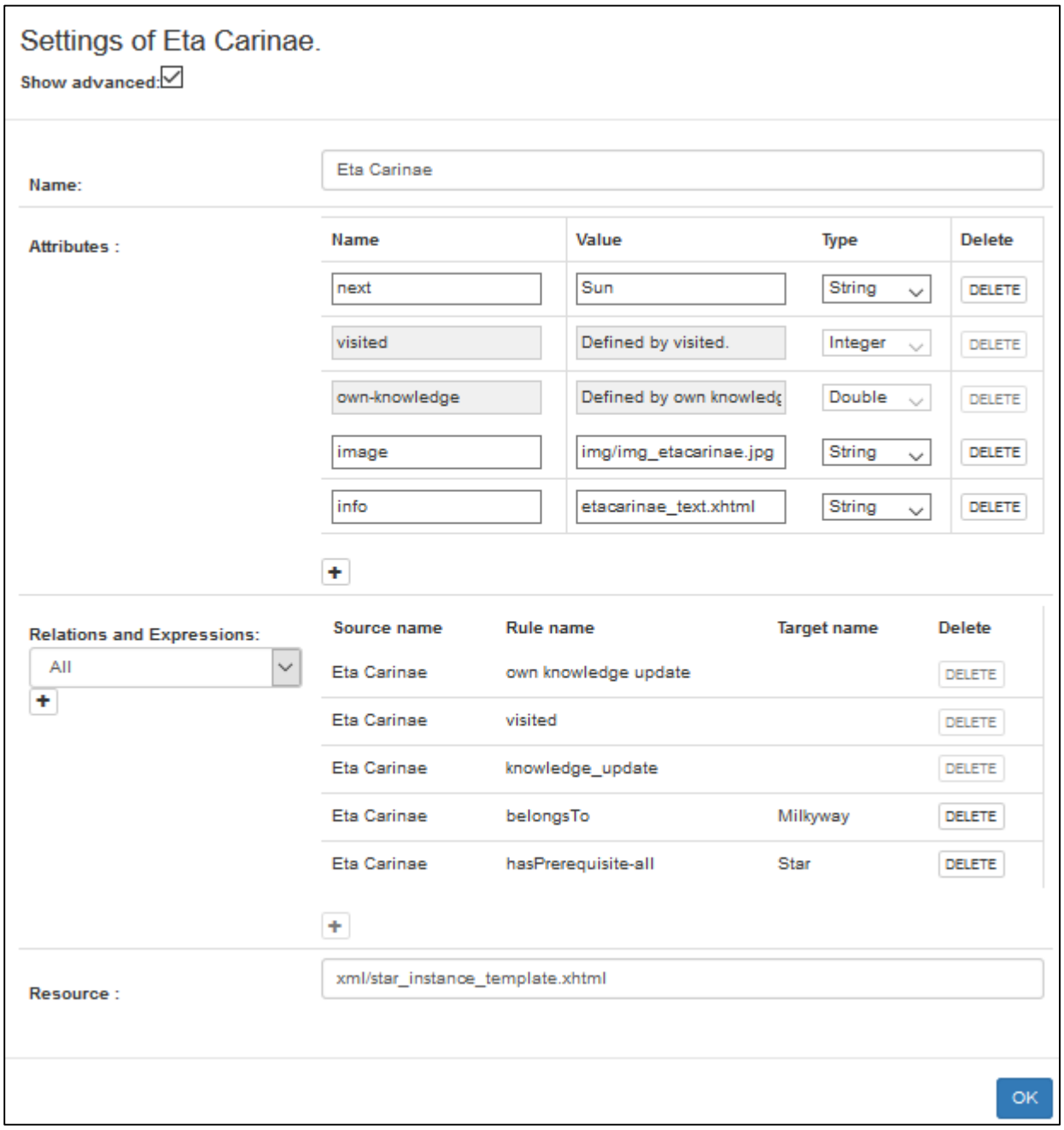

Figure A.9: The advanced settings screen

Appendix B

The ALAT User Interface Designs

This appendix contains some of the user interface designs as created during the design process of ALAT. They have been created by Yuexu Chen based on meetings between TU/e and “De Roode Kikker”. Note that these interfaces have been built with the purpose of AEH in mind. This is due to the combination of input given by both parties involved

Figure A.1 and *figure A.2* show the “step-by-step” and “overview” layout designs. As can be seen in *chapter 3*. These screens look a lot like the ALAT interface. In *figure A.3* a design is made of what it would look like to add a new concept to this hierarchy. At the time of creation, the templating system had not yet been thoroughly designed nor discussed. Therefore the interface designs have not included the concept blueprint selection as can be seen in *Appendix A*.

As mentioned before, the interface has been created with the purpose of creating an AEH authoring environment. Therefore the attributes and parameters visible in *figure B.4* and *figure B.5* are set according to the needs of “De Roode Kikker”. This is also the case for the adaptation rule design, in which only prerequisites can be added with a corresponding order. In ALAT the attributes, parameters and adaptation rules have been created to be more generic, as can be seen in *appendix A* as well as *chapter 3*.

Figure B.1: The step-by-step design

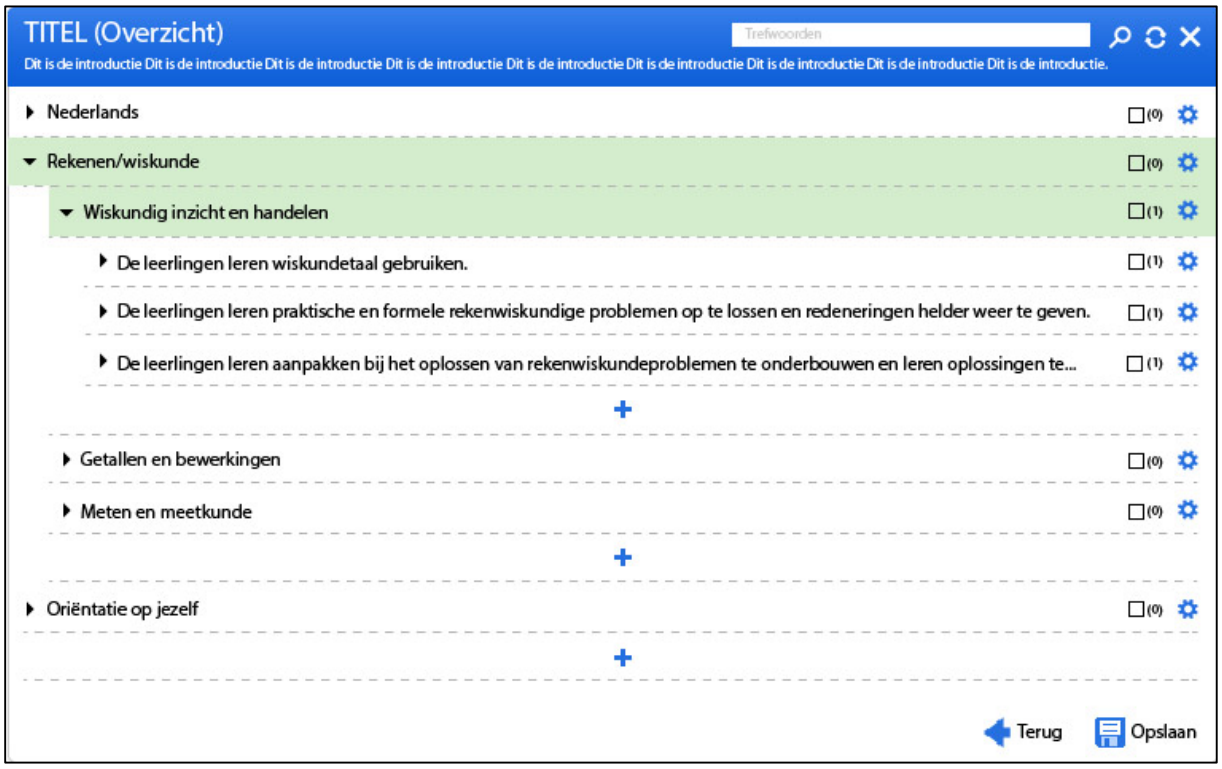

Figure B.2: The Overview design

Figure B.4: Adding a new item in the step-by-step view

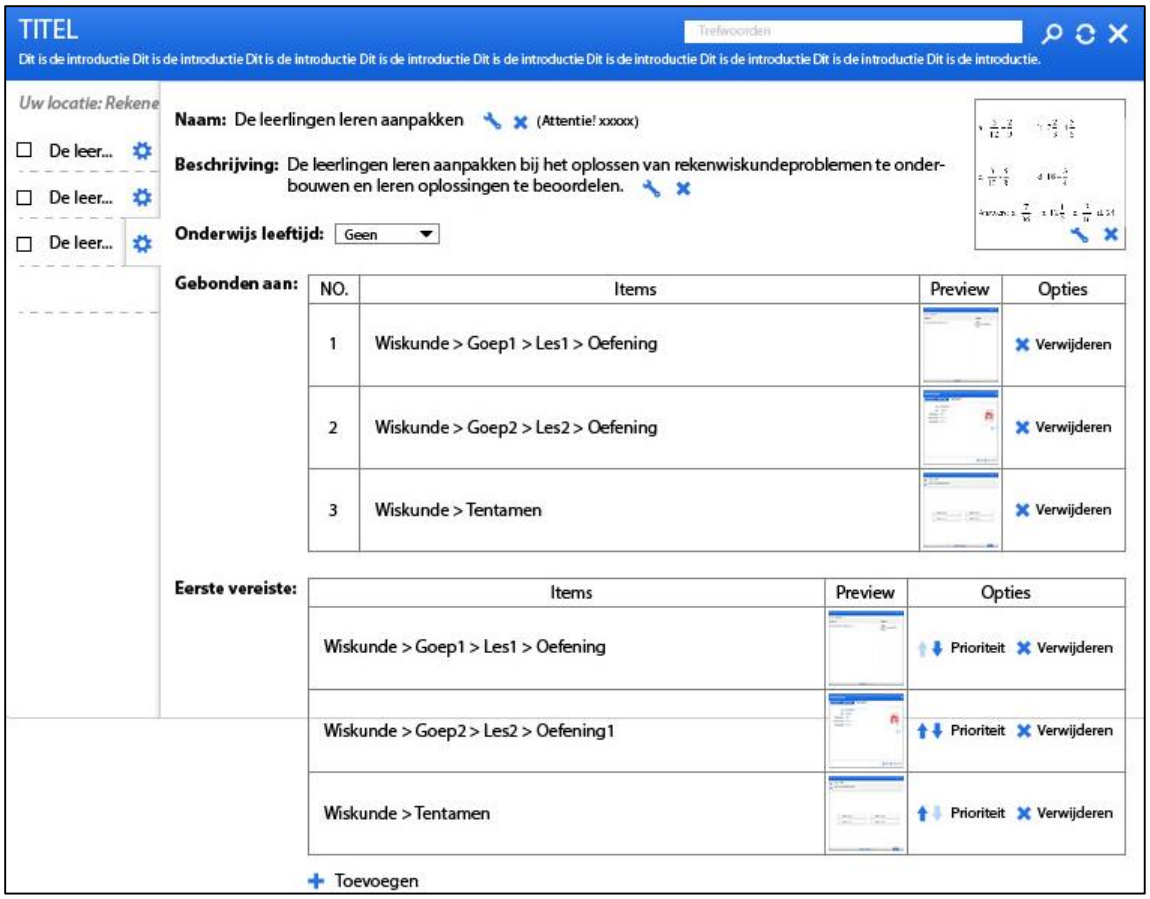

Figure B.3: The settings screen design:

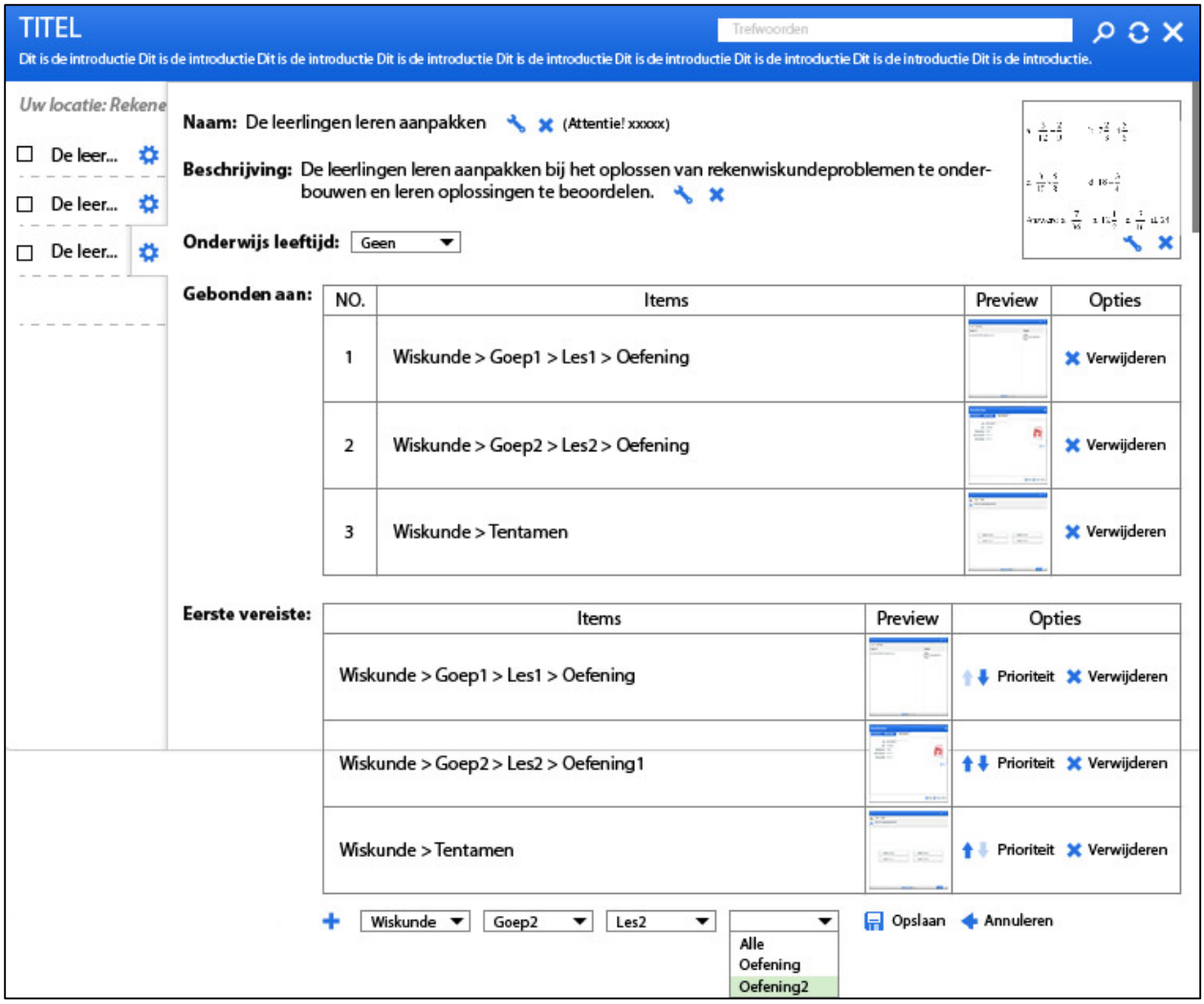

Figure B.5: Adding a prerequisite in the settings screen.

Appendix C

ALAT Architecture (Elaborated)

Figure C.1 shows a more elaborate version of *figure 3.7*. The relations between all controller, service and factory objects is expressed by means of dependency injection. This means that the object receiving the injection is able to access the injected object.

Example: the "*TreeController*" has 3 incoming arrows. These arrows lead to the "*GamService*", the "*ConceptService*" and the "*SessionService*". This means that these three service objects are injected into the "*TreeController*". The result of this is that the "*TreeController*" has access to all 3 of these service objects.

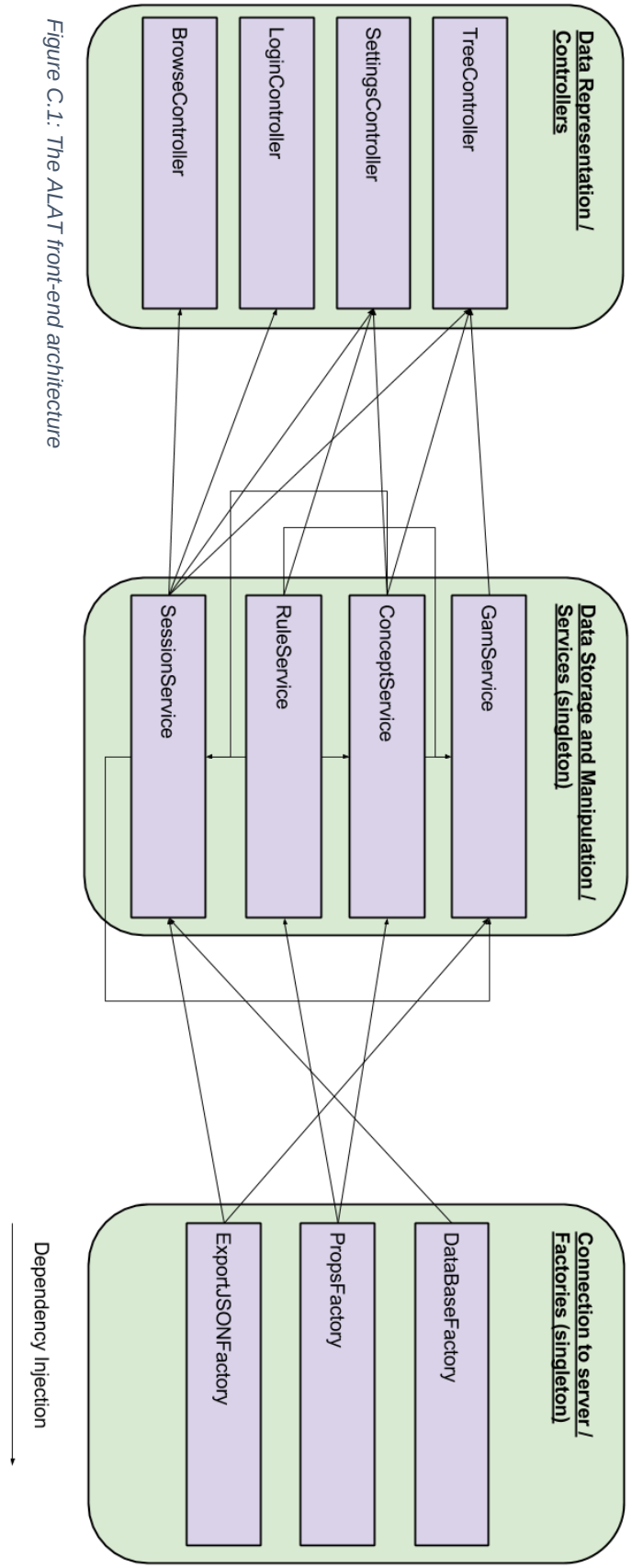

Figure C.1: The ALAT front-end architecture