GAM authoring tool
an authoring tool for GALE

Craenen, M.

Award date:
2017

Department of Mathematics and Computer Science
Web Engineering Research Group

GAM Authoring Tool

An authoring tool for GALE

Marc Craenen BSc

Supervisors:
prof. dr. Paul De Bra
dr. ir. Natasha Stash

Version 1.1

Eindhoven, November 2017

Abstract

For thousands of years, people have written on clay tablets, papyrus scrolls or just in plain books in order to share information and knowledge with other people. With the introduction of the internet, the size of the target audience as grown from just a couple of hundred to millions of people, each person with their own prior knowledge on a subject and needs. This has led to a growing demand for differentiating or adaptive hypermedia.

Until now, creating these adaptive hypermedia was only reserved for those who were either skilled in programming or had experience in using an adaptive engine like GALE. Although there were some authoring tools available that claimed to be easy to use, none of the authoring tools actually was easy to use for non-experts.

This paper introduces the GAM Authoring Tool, which is the newest authoring tool for GALE. During the development of the GAM Authoring Tool the ease of use has been one of the most important motives for each decision that has been made, which has led to an authoring tool that has overcome most of the problems other authoring tools in existence have.

Preface

During my master at the Eindhoven University of Technology I followed the course 'Adaptive web-based systems', for which we had to create an adaptive web application using GALE as adaptive learning environment. At that time there was an authoring tool called ALAT available, that could enable non-experts or novice GALE users help to create adaptive web applications. However, in my opinion, the tool was not that easy to use at all and missed several key elements that were needed to create an advanced GALE application.

Whether this was just because our application was too advanced for ALAT or that we were all skilled at programming, which meant that we probably thought far too complicated when using this tool, I leave in the middle. Eventually we ended up writing our own source code though, which was easier to learn instead of learning how we should use ALAT.

During my master's project I decided to write a new authoring tool for GALE that overcomes several issues that ALAT and other authoring tools in existence have. In this report you can read all about this newest authoring tool for GALE. It includes a detailed description about the most interesting decisions as well as newly introduced functionality. Although you are never ready developing the perfect application for everyone, I hope you will be both amazed and impressed after reading my work on the new authoring tool, called the GAM Authoring Tool.

Marc Craenen

Eindhoven, November 2017

Contents

List of Figures

List of Abbreviations

AHA! Adaptive Hypermedia Architecture

AHAM Adaptive Hypermedia Application Model

ALAT Adaptive Learning Authoring Tool

GALE Generic Adaptation Language and Engine

GAM GALE Adaptation Model

GAT GRAPPLE Authoring Tool

GDOM GALE Document Object Model

GEL GALE Expression Language

GRAPPLE Generic Responsive Adaptive Personalised Learning Environment

HTML HyperText Markup Language

JSON JavaScript Object Notation

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

Chapter 1

Introduction

During the past few decades the number of internet users has grown enormously. Where in 1995 less than one percent of the world population was connected to the internet, this has grown to almost 47 percent in 2016[9]. With this constantly growing number of internet users—who all have different intentions when using the internet—the need for hypermedia that is customised for each user has grown as well. Hypermedia that have implemented this adaptive behaviour are called adaptive hypermedia.

Next tho this growing number of internet users in general, digital devices are emerging quickly in various fields, which often require IT solutions adapted to the users' needs. In the Dutch secondary education for instance, there is a trend going on to introduce so called laptop classes at schools, of which one goal is to differentiate between the students. In order to achieve this goal, we need automated learning material that changes based on the students' performance in order to make sure that the system addresses the needs of the learner, instead of using a 'one size fits all' approach. In order to achieve this goal we need adaptive web-based learning platforms.

Until a couple of years ago, creating an adaptive hyperdocument was only reserved for those who where either skilled in programming or working with adaptation engines such as GALE[4]. This made it hard for people without these experiences or knowledge to create adaptive hypermedia that adjusted themselves to the users' needs.

Nowadays this problem has partly been solved by the introduction of so called *authoring tools*, which are tools that enable users to create adaptive hypermedia without any programming knowledge or particular knowledge in the field of adaptation. However, none of these authoring tools are quite ideal and all have some shortcomings, like being restricted to only textbooks or really complicated to use.

One of the most recent authoring tools for adaptive hypermedia is ALAT[1], which is an authoring tool that can be used to create adaptive websites that run on GALE[4], a generic learning environment for adaptive content. This tool has been published in 2016 and has overcome a lot of issues that users of previously published authoring tools had to cope with.

However, the tool has still some substantial issues which need to be tackled in order to make the tool really usable for everyone. This has led to the creation of a new authoring tool for GALE, called the GAM Authoring Tool. In order to prevent us from reinventing the wheel,

the tool has been developed with ALAT as starting point. That way we ensured that all useful functionality within ALAT is still available in the GAM Authoring Tool. During the development of the GAM Authoring Tool we kept the following three important requirements in mind, that served as guidelines for each decision that has been made:

- The authoring tool should serve the needs of both experts and non-experts, which means that inexperienced users should be troubled with entering complicated code anywhere, whereas this option is still available for users who want to use it.
- All actions and steps taken should be more or less intuitive, without reading a comprehensive user manual first. In other words: the tool should behave like the user expects the tool to behave.
- The tool should not only be able to compile the project into source code, but also be able to 'understand' the produced source code in order to detect (minor) changes to it and reflect these changes in the authoring tool.

In this paper you can read all about how these three goals are achieved. In Chapter 2 we first have a look at related work and material. Subsequently you can read about the development of the GAM Authoring Tool in Chapter 3, in which we discuss the chosen approach, choices that were made and new features that were added to the authoring tool. Chapter 4 provides some room for discussion and contains some ideas that could be included in future authoring tools, after which we conclude the whole project in Chapter 5.

Chapter 2

Related work

The newly created authoring tool produces, just like ALAT, code that can be used on the adaptive learning environment called GALE. In this chapter we highlight some of the most important features of GALE as well as ALAT, which will help to understand the rest of this thesis. An extensive literature study on these subjects can be found in Appendix B.

2.1 GALE

Just like developing an authoring tool that is suitable for everyone, creating an environment in which adaptive applications can run, turns out to be a challenge as well. In spite of that, in 2011 this has still succeeded with the introduction of GALE, the GRAPPLE Adaptive Learning Environment, which claims to be "a truly generic and general purpose adaptive hypermedia engine."[11] It therefore does not come as a surprise that when the GRAPPLE project[5] was concluded, GALE was renamed to Generic Adaptation Language and Engine.[10] In this section we describe some of the basic characteristics by explaining the main conceptual structures in GALE and describing some languages that could be used to provide GALE with input.

2.1.1 Three different models

Each course or adaptive application within GALE consist of several *concepts* that may have several *relationships* between each other. Designing this set of related concepts, which is called the *domain model*, is one of the first things you need to do when creating an adaptive application or course.

Figure 2.1 shows a small fragment of such a domain model. As you can see each concept is defined by its name, a set of *properties* and relations to other concepts. Whenever particular information about a concept is user dependant or more complex than a string, we need *attributes* (see Figure 2.2) to define this information, which all have an associated user model value. These attributes and properties can be inherited from other concepts as well by using the special *extends* relationship.

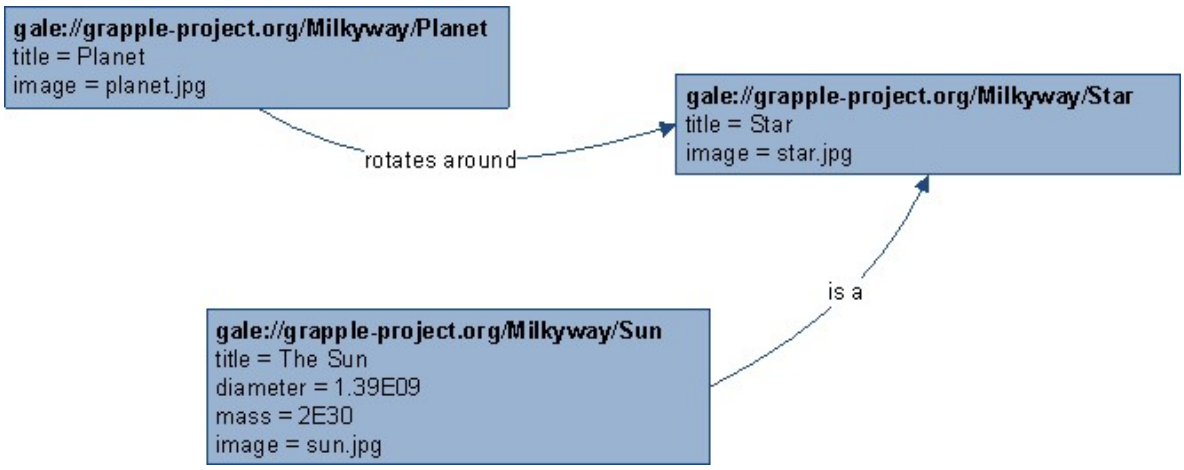

Figure 2.1: Graphical example showing a small fragment of a domain model.

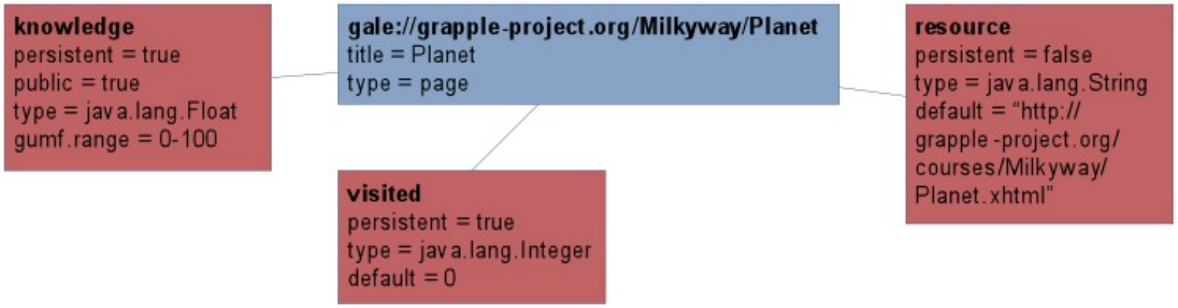

Figure 2.2: Another example showing a small fragment of a domain model.

Next to a domain model and unlike other hypermedia systems, adaptive hypermedia systems maintain a *user model*, which contains domain dependent and independent variables that are continuously updated. This user model keeps track of all information GALE knows about each individual user by keeping track of the values of all persistent attributes. In case an attribute is not persistent, the value of this attribute is recalculated whenever this is necessary, because these values are not stored in the user model.

Defining how the user model is updated is done by creating an *adaptation model*, which also defines how adaptation is done by combining information from the domain model and user model with the interaction of the user. In GALE there are two ways that tell the adaptation model what needs to happen: by using *events* or by adding adaptation to the content.

Events are pieces of code that can trigger updates to the user model by updating the values of the attributes that are stored in it. The actual code belonging to these events is stored in the domain model as properties of either a concept or an attribute. Adding adaptation to the content on the other hand, is performed by processors, which perform adaptation based on the information that is in the domain and user model.

2.1.2 GAM and GEL

Once you have defined the domain, user and adaptation model, you will have two write code that GALE interprets and turns it into an adaptive hypermedia application. This can be done in two formats: the GAM format or the GDOM format. Since only the GAM format is relevent for the GAM Authoring Tool, we will only treat this specific format in this section.

GAM is one of the formats GALE can use to store domain model information of a project. It is a textual formatted language that aims to be human readable and contains a list of concept definitions. The code below shows how such a concept definition is structured.

```
<concept-uri> {
    #<attribute-name>:<type> & "<default-expr>"
    #[<persistent-attribute-name>] "<default-value>" {
        <property-name> "<value>"
    }
    <property-name> "<value>"
    <property-name> + "<value>"

    ->(<relation-name>) <relative-concept-uri> {
        <concept-definition>
    }
}
```

Note that the text between the brackets needs to be replaced by suitable concept, attribute or property names and values. The code below is an example of how a piece of GAM code can actually look like.

```
Electromagnetism {
    ->(extends) _Electricity
    ->(extends) _Magnetism
    ->(parent) __root

    order "0"
    title "Electromagnetism"

    #resource = ‘~ return "[[=layout.xhtml]]";’
}
```

As you can see in the code above, the value of a property or attribute does not necessarily have to be a string, but can also contain an expression written in the GALE Expression Language, which is basically a piece of Java code extended with a special notation for using user model variables and domain model concepts, attributes and properties.

This GEL syntax is not only used within the GAM files you have to write, but can also be used when writing resources for an application in combination with the several modules that can be detected by the XMLProcessor. These modules are just XML tags that are custom made for GALE.

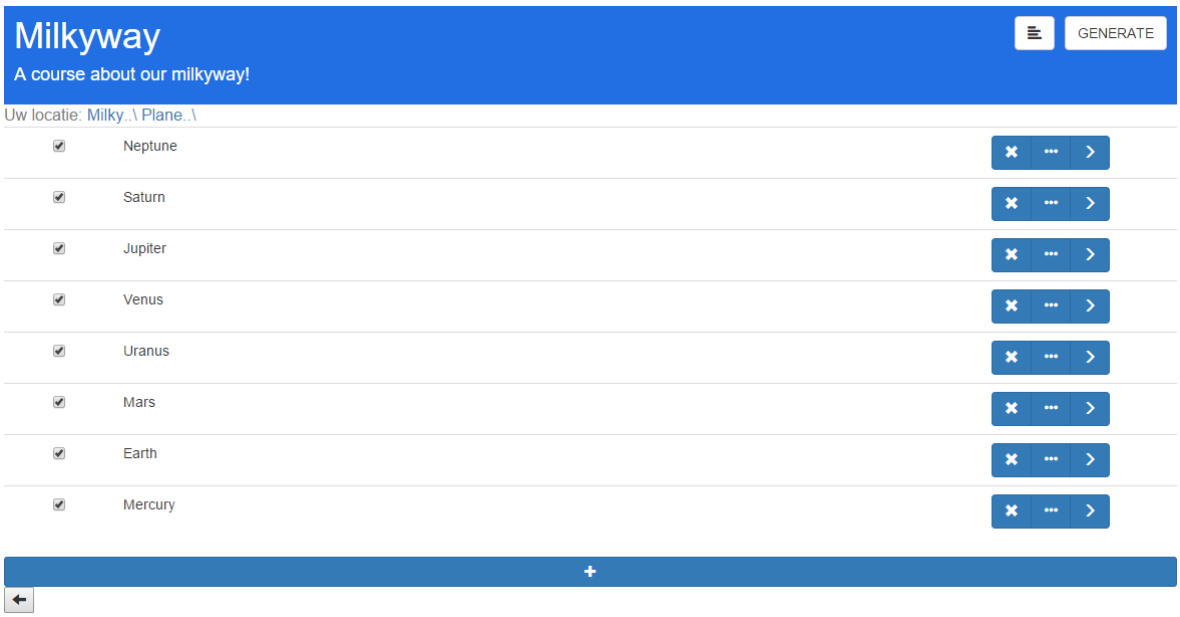

Figure 2.3: The step-by-step domain navigation in ALAT.

2.2 ALAT

As you might have noticed in the previous section, writing GAM code can be difficult if you are not familiar with writing GALE applications. In order to overcome this problem, several authoring tools have existed that help beginners or non-experts to create such GAM code, by presenting the user with a graphical user interface in which the domain and adaptation of a project can be created. ALAT is the latest authoring tool for GALE and has been developed as graduation project in 2016.[1] This section discusses some of the most interesting features in ALAT. More information about other authoring tools can be found in Appendix A and Appendix B.

ALAT has been introduced as a successor of GAT, which was an authoring tool for GALE that visualised the domain and adaptation model in order to make the authoring process less complicated.[1] However, it is hard to keep track of the overview when the number of concepts and relationships grows and does not have the desired functionality and usability either. ALAT has been developed to overcome these problems. Furthermore, it has been designed to be a generic authoring platform that helps users to easily create adaptive hypermedia without any extensive knowledge of the GAM format or even GALE.

The one thing that stands out in comparison with other other authoring tools is that ALAT does not have the graph-like domain and adaptation model representations that both GAT and AHA![6] had. Instead of this, ALAT comes with a tree-like structure in which concepts are connected through parent relations, which incidentally is the same hierarchy that is used by the static-tree-view in GALE, which usually shows up as the menu of the created hyperdocument.

ALAT supports two ways of viewing this hierarchy. The first one, that can be seen in Figure 2.3, provides the user with a step-by-step domain navigation in which the number of concepts on

Figure 2.4: The domain navigation in ALAT as collapsible list.

the screen is kept to a minimum in order to prevent the user from having an overload of information. The second one on the other hand, that is shown in Figure 2.4, provides the user with a collapsible list, which is designed to be fast to navigate.

Next to this concept screen containing the domain hierarchy, ALAT has a second screen which contains all information concerning a concept that has been selected in the domain hierarchy. In this screen the user has the possibility to add, edit or remove properties, attributes, adaptation rules and non-pedagogical relations that are coupled to that concept. Figure 2.5 shows a printscreen of this screen.

Once a user is finished creating the domain and adaptation model in ALAT, ALAT has to generate a GAM file that can be deployed on a GALE server. This is done by pressing the 'Generate' button in ALAT. By doing this the user is ensured that the complete ALAT project is turned into a GAM file, which is then copied into a directory together with two XHTML files that can be used for adding content to a concept. This directory can then be visited by a GALE server, which takes care of reading these GAM and XHTML files and turns them into an adaptive application.

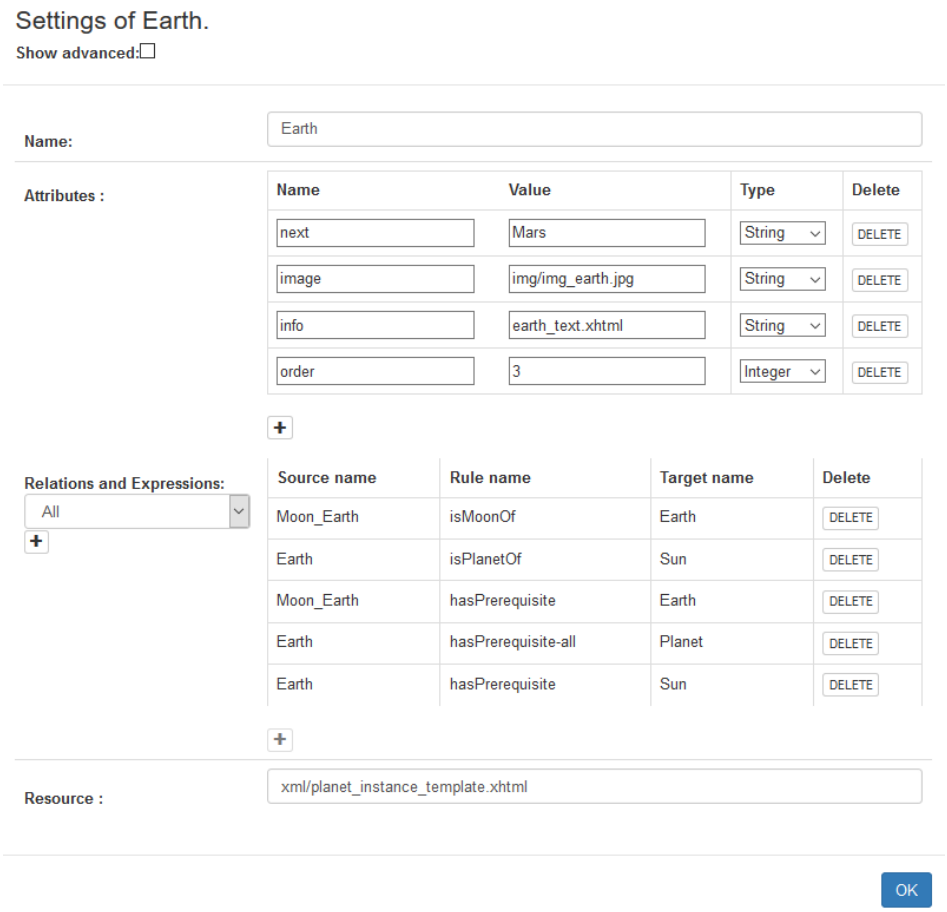

Figure 2.5: Screen containing all information related to a concept.

Chapter 3

GAM Authoring Tool

As mentioned in the in seminar project (see Appendix A) that preceded this master's project, the existing authoring tools for GALE and for other adaptive hypermedia environments have a number of problems, which make those tools difficult to use. Hence the goal of this project: to create a new authoring tool for GALE that overcomes most of these problems, by following the three guidelines described in Chapter 1. Next to overcoming these problems, new functionality had to be included as well in order to make the tool as user friendly as possible for both experienced users as well as users that are new to the concept of authoring adaptive content.

3.1 Used techniques

The GAM Authoring Tool is partly based on ALAT, since ALAT had already overcome a lot of problems that previous authoring tools for GALE had. It is however still created from scratch, because one of the requirements for the GAM Authoring Tool was that it is integrated with GALE, which requires the tool to be written in the programming language Java. Since GALE—which is written in Java 6—has some troubles while running in a Java 8 or higher environment, we decided to write the GAM Authoring Tool in Java 7, with which GALE does not have any problems.

Since GALE runs as a web application in a Tomcat server, it seems almost obvious that the GAM Authoring Tool uses HTML5 for the user interface. This ensures that all decent browsers are able to render the pages in more or less the same way, without installing any plugins, extensions or add-ons. In order to make some parts of the tool interactive, we used the cross-platform JavaScript library jQuery.

Furthermore, ALAT was not responsive, which makes it very difficult to work with ALAT on mobile phones and tablets. To overcome this problem we decided to use the Bootstrap, which is a free and open-source front-end web framework for designing websites and web applications. Those who have carefully inspected ALAT might notice that ALAT used Bootstrap as well in its HTML output. However, ALAT used only small parts of the Bootstrap framework, which did not lead to a responsive design.

3.2 Approach

Before starting on implementing the GAM Authoring Tool I had to investigate what trouble a non-expert on adaptive hypermedia could run into when using ALAT. I had already created a list on my own during the seminar project, but this list was too incomplete to already start working on the project. Therefore I let several of my family members try to recreate the Milky-way example[1] that comes with every GALE installation. This has led to the following list of possible improvements to ALAT:

- Besides compiling the created project in the GAM Authoring Tool into GAM code, it should also be possible to read a GAM file and create a GAM Authoring Tool project based on this GAM file.
- It takes quite some mouse clicks to achieve minimal action in ALAT. For example, after signing in you need to select a project you want to work on. Intuitively each test person double clicked this project, but nothing happened. After selecting you project, you have click 'load' in order to load the project. The GAM Authoring Tool should take this number of mouse clicks to a minimum in order to make the tool more user friendly.
- The visualisation of the concept hierarchy in ALAT is not consistent. The shown visualisation at the overview differs completely from the visualisation you will get to see when you want to add relationships. This, in combination with buttons that do not get obscured nor non-clickable, leads to a way of adding relationships which is counter intuitive. Next to this, the tool is inconsistent in other areas as well. The button to go deeper into the concept hierarchy looks for instance quite different from the button that brings you back one level.
- Besides the concept hierarchy at the 'home page' of ALAT, there is no visual overview for relations, which makes you click every concept when you want to find out how often a certain relation is used.
- After selecting a project, there is no way to get back to the previous page and select a different project. There is not even a log out button in order to sign in again and select another project.
- Textual explanation of options and buttons is rarely shown, which makes it hard for non-experienced users to understand what they have to do. Sometimes there are tooltips available, but they are always shown at the centre of the screen, instead of the place you expect them to appear.
- When creating a concept that is a child of another concept, it is not immediately clear how this works. It actually took some family members several minutes before they found out that they could change the view to one that makes it easier to understand. The reason it took several minutes was that the check box before the parent concept was selected and they all assumed that clicking the plus sign would result in adding a child to that concept. This is because this functionality is again counterintuitive and the tool lacks an explanation about what the check boxes do.

[1] `http://gale.win.tue.nl/gale/concept/gale://gale.tue.nl/course/milkyway/Milkyway`

- You cannot edit or create templates in ALAT itself, nor is it possible to see what a template looks like, without creating a concept first.
- It is not possible to move a concept to another location in the concept hierarchy. You have to delete and recreate the concept in order to achieve that.
- ALAT contains several bugs. For example: ALAT allows that two concepts have the same name. This is certainly not allowed in GALE, nor is this desirable. Another thing that does not work properly is (un)publishing a concept. When you (un)publish the check boxes of all the descendants are flipped, with the purpose that these descendants are (un)published as well. But when you unpublish a concept, the already unpublished descendants suddenly become published.
- The order in the menu of a generated project differs from the order shown in the concept hierarchy of ALAT.
- When adding a relationship to a concept, you have to select the type of relationship first, after which you need to click on the plus sign followed by selecting the target. This is not only counterintuitive, but also very confusing since there are three plus signs shown in the same page, which all three have an unclear purpose. When you have managed to add a relationship to a concept, it is not immediately visible which relationships are outgoing and which ones are incoming and if a concept does not have any relationships at all, it just shows an empty area, without giving any feedback about what this empty area means or does.
- Some relationships cannot be added to certain concepts, but the used colour is just a little different from the rest of the tool, which makes it hard to see which relationships cannot be added. Sometimes there is not even any distinction between the colours. For example: If you try to add a relationship with the same target concept as root concept, the colour of the concepts are all black, even though you cannot select a target concept that is identical to the root concept. After adding a relationship the select-bar does not move, which gives the impression that nothing actually happened.
- It is not possible to work with several authors on one project.

After the inventory of all shortcomings of ALAT, the work on the GAM Authoring Tool could begin. During the actual development of the tool, weekly meetings with my graduation supervisors ensured that we kept continuously on track. During these meetings several other improvements and suggestions were named, which led to several interesting choices, which you can read about in the next section. The next sections describe the most important choices that were made, as well as the main differences between ALAT and the GAM Authoring Tool.

3.3 Choices

3.3.1 Templating in the GAM Authoring Tool

One of main differences between ALAT and the GAM Authoring Tool is the way of creating templates for concepts. In ALAT you are required to write one ore more JSON files in order

to create templates for newly created concepts, where there are two types of templates:

1. Blueprints: the blueprints are a set of basic concept types which are selected as a basic structure for new concepts, i.e. it contains a set of attributes and adaption rules that the new concept has to adhere to.
2. Pedagogical relation definitions: The relation definitions on the other hand define all the rules and relations that exists within ALAT. Each time you want to add a new relationship between two concepts, the type of this relationship has to be defined in this JSON file before doing this.

It does not come as a surprise that writing JSON code could be difficult for users without any background in programming. In order to enable all users to create such templates the GAM Authoring Tool makes a distinction between *concepts* and *concept types*. A concept type can define several attributes that each concept that extends this concept type inherits. This makes writing complicated JSON blueprints unnecessary.

However, there are two things these blueprints in ALAT supported, which are not possible in the GAM Authoring Tool anymore. The first thing is that ALAT supported is a default concept name. Since each concept needs a unique name anyway, dropping this particular functionality does not come with any drawbacks. Next to this, the blueprints could make sure a concept had a couple of default properties as well. Since properties are not extended by default in GALE, this does not work in the GAM Authoring Tool anymore. There are ways to include this functionality in the GAM Authoring Tool as well, though, for instance by giving properties in a concept type a check box that determines whether this property should be inherited or not.

Multiple concept types

With using concept types instead of using blueprints comes the decision whether or not to allow the possibility to extend multiple concept types. Intuitively it makes sense to restrict a concept to extending at most one concept type, similarly to extending classes in programming languages like Java en C#, because extending multiple concept types could result in conflicts. These conflicts will occur when for instance both concept type A and B have an attribute called ‘advanced’. When this attribute has been given the value ‘true’ in both concept types, then nothing is wrong, since a concept that will extend both A and B automatically inherits the attribute ‘advanced’ with value ‘true’. Assume however that for concept type A this attribute has the value ‘true’, while for concept type ’B’ this attribute has the value ‘false’. This will result in a conflict, because it is unclear which value the concept should inherit. Even worse, the attribute could be of different data types in concept types A and B.

GALE itself does however allow the possibility to extend multiple concept types and it could actually be useful in several cases, so we have chosen to allow extending multiple concept types as well provided that we could overcome the problem of inheriting multiple attributes with the same name. There are several ways to overcome this problem, three of which we have taken under consideration:

1. Choose the value that GALE uses.
2. Choose the value based on the name of the concept type.
3. Choose the value based on the order of extending the concept types.

The first and most logical option to tackle this problem is the one that is almost impossible to implement. This is due to the fact that GALE uses the value of the concept type that comes first in its collection with all concepts of the application, but there is no way to actually tell which one this is, because this collection is implemented as a set, which does not have a deterministic order according to its nature. Although this order is actually deterministic in Java, there is no way to tell what this order will be, so this option cannot be used.

The second option will probably give the user an headache when the user renames a certain concept types, as this influences the way of inheritance for every concept that extends the renamed concept type.

This leaves us with the third and last possible option, that chooses the inherited value based on the order of extending the multiple concept types. Although this option has some drawbacks as well, they are easier to overcome and easier to explain to the end-user. Keeping in mind that the GAM Authoring Tool should be used by non-experienced users as well, it is this option that is implemented in the GAM Authoring Tool.

In order to implement this option the user should have the possibility to change the order of extending the different concept types without removing and adding a concept type again, as this could result in removing relationships as well, which you can read more about in Section 3.3.3. Hence it has been made possible to change the order of extending by dragging-and-dropping a concept type in its correct position. Once an attribute is inherited twice, but each time with a different value, the value of the last extended concept type will be chosen.

Since there is no way to predict in which order GALE extends the multiple concept types, we had to find a way to make sure GALE actually takes the correct values. In case of an attribute that is used in only one concept type, we do not have to do anything special, since GALE just uses the only available value for this attribute. However, when multiple concept types have an attribute with the same name, we have to take care of this problem by explicitly including the correct attribute value in the produced GAM code for that concept.

Although this solves the problem of inheriting multiple attributes with the same name, it introduces a new problem as well. When we explicitly include the correct attribute value in the GAM code for that concept, we cannot tell that this attribute is inherited by just looking at the GAM code, since it will look like that attribute belongs to that particular concept. This gives us problems when turning the GAM code back into project code. Nevertheless, we have been able to solve this problem by adding comments within the GAM file that can be detected by our GAM code parser, which will then skip the inherited attribute.

3.3.2 Code coupled to concept (types) and attributes

As mentioned in Section 3.3.1 ALAT needs a JSON file containing all rules and relations that exist within ALAT. These rules can either be unary or binary rules and these binary rules do not directly correspond to anything in the GAM format. These binary rules behave like relationships, but are actually translated to a chain of expressions that determines the value

of an attribute, as you can see in Example 3.3.1. It is not difficult to see that this results in complicated GAM files which are difficult to read once a concept has multiple prerequisites.

Example 3.3.1. Assume the binary *AND* rule 'hasPrerequisite' is defined with the code:

```
${%target%#knowledge} > 0.8
```

And let concept A and B be a prerequisite for concept C. ALAT will then produces the following GAM code and will put this in the concept definition of concept C:

```
#suitability:Boolean = "true && ${A#knowledge} > 0.8
                        && ${B#knowledge} > 0.8"
```

Since the GAM Authoring Tool not only needs to create GAM code, but also needs to be able to 'read' it and turn it back into a GAM Authoring Tool project, the project needs to resemble the GAM file as much as possible, without becoming too complicated to use for non-experienced users. Example 3.3.1 makes clear that ALAT does not work like this, because a rule is added as a relationship within ALAT, but is compiled into an attribute when producing the GAM file.

In the GAM Authoring Tool you will need to use relationships in order to achieve this. The GAM Authoring Tool makes a difference between *relations* and *relationships*. The first one is a certain type of relation, while the latter is an instance of this relation. In order to add a relationship to a concept, you will need to define the type of that relationship, i.e. the corresponding relation first. Instead of doing this in a JSON file like ALAT, you can do this using the tool, in which you will also see an overview of the already defined relations, as you can see in Figure 3.1.

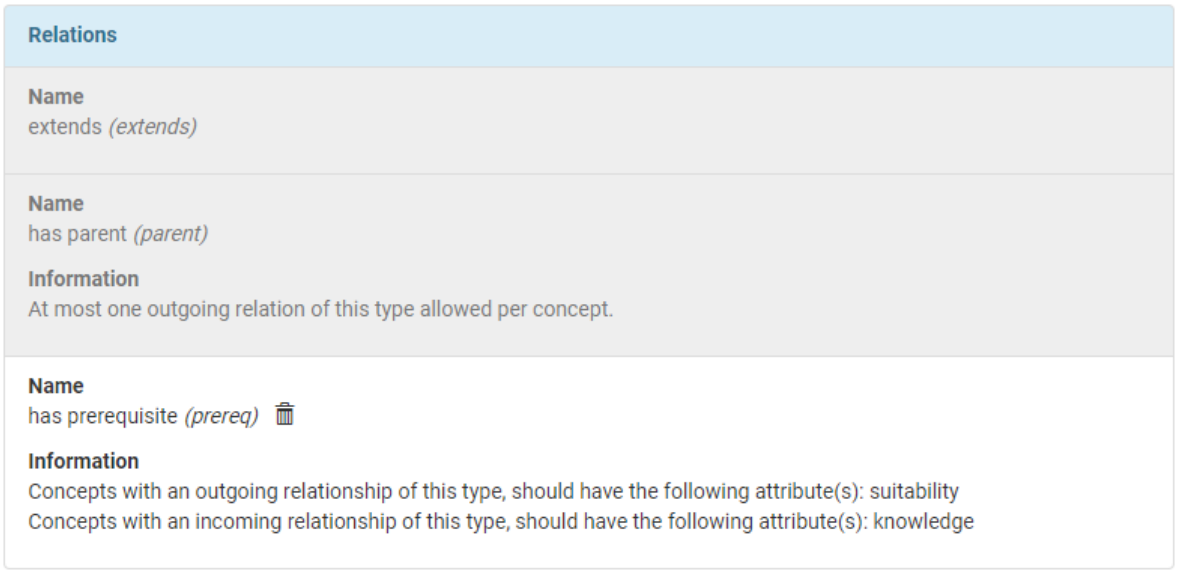

Figure 3.1: Example of added relations to a project.

In Figure 3.1 you can see that—next to the built-in relations *extends* and *has parent*—there exists a relation called 'has prerequisite'. You can add a relationship of this type when the concept that needs to be a prerequisite has an attribute called 'knowledge' and the concept

that requires the prerequisite has the attribute 'suitability'. In Section 3.3.3 you can read why this is done, but it ensures that all concepts that serve as prerequisite have an attribute called 'knowledge'.

When using the authoring tool in the advanced mode, you then just have to add the following attribute to a concept. Note that it is possible to add this attribute to a concept type as well, which makes sure that it is inherited by all concepts that (in)directly extend this concept type:

```
#suitability:Boolean = "Double.valueOf(Collections.min(Arrays
                        .asList(${=>(hasPrerequisite)#knowledge})
                        ).toString()) > 0.8"
```

This piece of code has the exact same effect as the code in Figure 3.3, but comes with the advantages that the structure almost matches one on one with the structure used in the GAM code of some existing applications and the produced GAM code is easier to read and understand when the number of prerequisites keep increasing, since this will remain unchanged. Note that the above code is still not ideal to read, but with the addition of just one more attribute called 'known' to all prerequisites that checks whether the knowledge of that concept is more than 0.8, you can change above code into:

```
#suitability:Boolean = "and(${=>(hasPrerequisite)#known})"
```

Furthermore, using these kind of GEL expressions also has the advantage that you are already using the same expression language that you will need to use when you are writing adaptive XML pages. Hence you will not have learn multiple programming languages in order to make full use of all of the possibilities the GAM Authoring Tool has to offer.

3.3.3 Restrict relationships to attributes

Another difference compared to ALAT is that in the GAM Authoring Tool some relations require certain attributes on the target and source concepts. As explained in Section 3.3.2 it is possible to use relations in order to create prerequisites for certain concepts. But the code that actually enforces this prerequisite assumes that the target and source concepts have the attributes knowledge and suitability respectively. In order to make sure this is actually the case, the 'has prerequisite' relation in Figure 3.1 can only be added between concepts that meet this requirement.

An alternative way for ensuring that there is no invalid GAM code would be to restrict certain relations to one or more concept types. The main advantage of this approach would be that you will always know for sure that a concept has the required attributes, since they are inherited from the extended concept type. The big disadvantage and the main reason that this approach has not made the final version of the GAM Authoring Tool is however, that you will need to create multiple extra concept types that only serve the purpose of ensuring that certain attributes are inherited by the concepts that need specific relations. This could quickly lead to a complicated and difficult to read project, where you lose track of the main structure.

There is actually one solution that does not bother the user of the tool with structures that are hard to understand, namely by analysing the code and detecting whether this code can actually be executed. Although this seems like an easy solution, it is actually harder than it

looks, as there are no simple ways to guarantee confluence.[12] By this we mean that when a certain event will trigger some other events and these events will trigger other events again, we cannot always guarantee that the order in which these events are executed will not influence the outcome. Even worse, there are no simple ways to guarantee termination either, as it is computational intractable to guarantee this.

3.3.4 Unpublishing a concept (type)

As mentioned in Section 3.2, ALAT has the possibility to publish or unpublisch certain concepts, which determines whether a concept will be included in the next release. This feature might come in handy when the author has created several concepts, but does not want to publish them yet. Example 3.3.2 provides a good example why a user might want to unpublish a certain concept or concept type.

Example 3.3.2. Assume you are a teacher and want to create an adaptive web-based textbook, containing several different chapters on various topics. Normally you would cover the whole book, but this year you decided to slightly change the curriculum, so you skip the last two chapters. In order not to confuse the students you decide to unpublish these last two chapters, so it is clear that they will not have to learn these.

With the introduction of concept types in the GAM Authoring Tool, the way of (un)publishing a concept or concept type had to be determined as well. Although this seems like an easy question to answer, it is not, since non of the possibilities seems completely right. In the next sections we explain what should happen to a concept when its parent or one of its concept types is unpublished, what should happen to relations between that concept and another (published) concept and why we decided to implement this particular action instead of the alternatives.

Unpublished parents

The first thing that needed to be determined was what should happen when the parent concept of a concept becomes unpublished. On the one hand it makes sense to unpublish the child concept as well, because if you do not want to cover a certain chapter, you will probably skip all sections in that chapter as well. Have a look at for instance Figure 3.2. If you want to unpublish the concept 'Atom', you want to unpublish the concept 'Neutron' as well, because you need to know what a concept is, before learning about neutrons.

On the other hand, however, still being able to visit the children of the unpublished concepts makes sense as well. There are good examples to support this action method. Have a look at Figure 3.2 once again and suppose you have learned in physics class that in order to have an electric current, you will need a flow of electric charge, which is usually carried by moving electrons. Knowing how an electric current works, requires you to know something about electrons. However, you are not required to know about atoms to be able to understand this concept. So it would be fine to unpublish the concept 'Atom' without unpublishing the concept 'Neutron'.

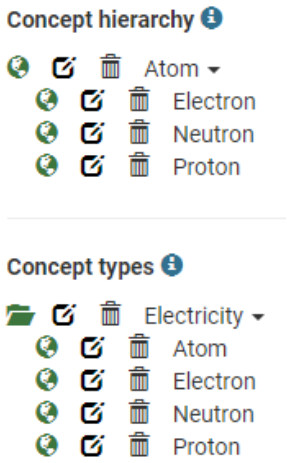

Figure 3.2: Example concepts and concept types with atoms.

So in short it had to be decided whether being able to read a certain concept has an effect on the visibility of its children. For the GAM Authoring Tool we have decided that it is not possible to publish a concept, when one or more of its ancestors are unpublished. The reason for this is that it seams more natural to read a complete chapter instead of only a couple of sections within a chapter.

Furthermore, there was no clear solution for what should happen to the menu. Once you remove a concept from the hierarchy, but still allow users to visit the children, then *where in this hierarchy should you show the children? Should the children go up one level in the hierarchy? Should the children disappear from the hierarchy? But if you do this, what if no other page refers to this concept?* Allowing visiting children of unpublished concepts raises many more questions that it solves problems and with the thought in mind that the tool should be easy to use and understand for non-experienced users, children of unpublished concept are unpublished as well.

Unpublished concept types

A similar problem arose when deciding what should happen when a certain concept type is unpublished. As mentioned in Section 3.3.1 a concept type could serve as a certain template for other concepts and concept types. You could give the user the possibility to disable this template by unpublishing it, which makes sure that all concepts and concept types that extend the unpublished concept type will not conform to that template anymore.

Implementing the way of unpublishing this way may lead to unwanted results, as concept types are not only used as a template, but also as categories. By unpublishing such a category the user expects that all concepts that belong to this category will be unpublished as well.

This again gives us problems, though, which usually arise when a concept extends multiple concept types, as shown in Figure 3.3. Here you can see that the concept ‘Electromagnetism’ extends the concept types ‘Electricity’ and ‘Magnetism’, that serve as a category. However, ‘Electromagnetism’ belongs to two categories. By unpublishing one of those two concept

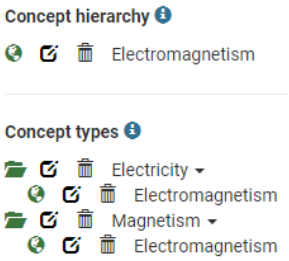

Figure 3.3: Example concepts and concept types with electromagnetism.

Figure 3.4: Legend showing how to recognise an (un)published concept (type).

types, you expect that all concepts that extend that concept type are unpublished as well. But you expect that all concept that extend a published concept type are published as well, which results in in conflict. In this case the GAM Authoring Tool will still unpublish the 'Electromagnetism', because we assume that you are not able to understand the concept of electromagnetism if you have never covered any topic on either electricity or magnetism.

In order to make it immediately clear to the user whether a certain concept or concept type is published or not, we provided the user with a small legend (see Figure 3.4 at the bottom of the page. This legend can be used to quickly determine if and why a concept or concept type is (un)published.

Relationships

A last important—but quickly answered—question was what should happen to the existing relationships between an unpublished concept and a still published concept. For these situations we have decided that these relationships will not be included in the GAM-code as well. The alternative does namely easily lead to prerequisites that are never fulfilled, because you cannot visit the prerequisite, since it is unpublished.

3.4 New functionality compared to ALAT

3.4.1 Relationship overview

One of the functionalities that was available in the GRAPPLE Authoring Tool, but had been omitted in ALAT was the visual overview of existing relationships. Although this functionality had the problem that it became more difficult to read as the number of concepts and rela-

tionships grew, we reimplemented this functionality in the GAM Authoring Tool nevertheless, since ALAT does not come with an overview of all relationships at all, which is even worse than a diagram that is far too cluttered.

In Figure 3.5 an example of this visual overview can be found. By pressing the gear wheel in the top-left corner you can select which relationships you want to visualise. This may come in handy when you only want to investigate one or more types of relationships and do not want to be bothered with all the other relationships. Next to this you have an option to view concepts that do not have one of the selected relationships. In this way you can quickly view whether or not there are concepts without the selected relationships in your project.

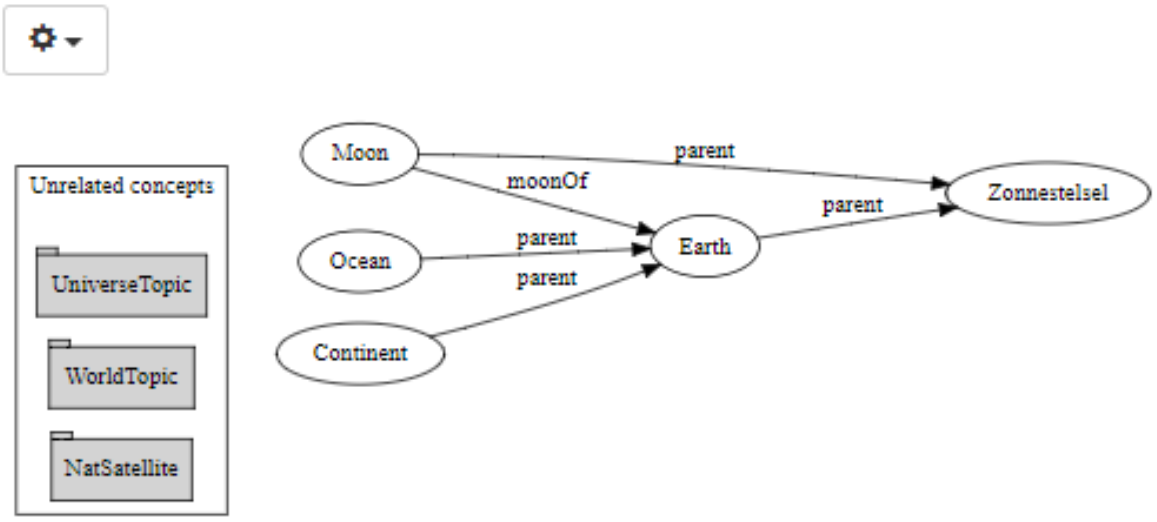

Figure 3.5: Example of the relationship overview.

3.4.2 Changing the hierarchy

When a user accidentally created a concept somewhere in the hierarchy where it does not belong, the user can change this in the GAM Authoring Tool by changing the parent concept of this concept. This functionality, which gives the user the possibility to move a concept to another place in the hierarchy, was not available in ALAT. When a user incorrectly inserted a concept in a certain place, the user had to remove the complete concept, followed by recreating the complete concept again in the correct place. Implementing this feature in the GAM Authoring Tool ensured that the user does not have to redo a lot of work when the user wants to change the hierarchy of the project for any reason.

3.4.3 Multiple authors

Sometimes it is desirable to have multiple co-authors working on a project. While this was already possible in the GRAPPLE Authoring Tool, ALAT had dropped this functionality. Because it seemed like a nice feature to have, it is reintroduced again in the GAM Authoring Tool.

3.4.4 Reading GAM code

Adding the possibility to read GAM code is the most important new feature. Sometimes it would be useful to not only compile a GAM Authoring Tool project into GAM code, but also GAM code into a GAM Authoring Tool project. This could come in handy when a user has authored a project in the GAM Authoring Tool, but made some small mistakes or wants to fine-tune some of the code. Instead of modifying these attributes or event codes in the tool, it is much faster to make those changes in the GAM file. Once the user adjusted this GAM file, these changes can be reflected in the GAM Authoring Tool.

While in theory this could work with every GAM file, it is only meant to used in GAM files that just have minor changes compared to the GAM file that was created by the GAM Authoring Tool. This is due to two reasons. The first one is that the GAM parser within the tool only allows a single GAM file to be uploaded, which means that GAM files that contain references to concept contained in another GAM file are not supported and therefore cannot be interpret by the GAM Authoring Tool.

The second reason is that several problems arise when multiple concept types are extended, as mentioned in Section 3.3.4. To resolve this problem, the GAM Authoring Tool adds comments at various places in the GAM file, to ensure that some code within the GAM file is skipped when trying to parse it. When a GAM file is presented that does not contain any comments, the tool cannot guarantee that the processed GAM file completely reflects the given GAM file. In case the tool is not sure that the GAM file is processed properly, the GAM Authoring Tool will notify the user that some discrepancies might occur.

Chapter 4

Discussion

As mentioned several times in this thesis, the GAM Authoring Tool is not perfect as some features and choices restrict the user to (re)create any GAM file. This chapter describes some possible improvements on the GAM Authoring Tool and describes the vision I have about adaptation in the feature.

4.1 Improvements on the GAM Authoring Tool

During the development of the GAM Authoring Tool we came up with numerous ideas that would be useful for the GAM Authoring Tool to have and that were not (yet) present in ALAT. Mostly due to the finite amount of time available for this master's project, not all of these functionalities are included in the final version of the GAM Authoring Tool. In this section some of these functionalities is described.

When visiting a project using GALE you have to know the concept you want to visit first. So if you want to start at the beginning of the project, you need to specifically mention which concept this is, which does not make any sense if you never visited the project before. It would be nice to have some kind of *root concept* that is visited automatically when no specific concept is given, which may even depend on the user's knowledge.

As mentioned in Section 3.3, extending the GAM Authoring Tool in such a way that properties can be extended as well, would be a useful feature to have. GALE does not extend properties by default, but this can be achieved by defining an ~extends attribute on a relationship for each property that should be inherited. By adding a check box that can be selected when the user wants to inherit this property, this functionality can be included in the tool as well.

The last improvement that would be helpful for the user to have, is that the user receives a notification when a prerequisite cannot be achieved. As already explained in Section 3.3.3 it is actually intractable to detect this, since we need to guarantee confluence and termination in order to do this.[12]

Next to all possible improvements described above, it would also be nice to have the tool tested by numerous people, each with different intentions, knowledge about programming and knowledge of GALE. By having the tool tested each time a major change has been made, we can make sure that the tool fits the needs the highest number of people as possible.

4.2 Adaptation in the future

While *creating* the domain and adaptation model is the main and only feature of the GAM Authoring Tool at this time, it would be nice to include functionality to *maintain* the adaptation model as well. Assume that a concept is visited often whilst the prerequisites are not met yet, then there is probably something wrong with the adaptation model. Or assume that users tend to visit concept A first, then go to concept B and switch to concept A again, even though concept B is not a prerequisite of concept A. In these cases, the tool could show a notification to the user that it suspects that the adaptation model needs some fine-tuning. It would be even better if the GAM Authoring Tool could detect which prerequisites it should add or which piece of content needs to be copied from concept B to concept A in order to make sure the user has the correct prior knowledge. However, in order to make it possible to modify the adaptation model automatically, GALE should keep track of numerous extra bits of information, like the order all concepts are visited.

Although GALE has been renamed from 'GRAPPLE Adaptive Learning Environment' to 'Generic Adaptation Language and Engine', it still suits hypermedia that are meant to teach the reader something about a certain topic best. It is not easy to use GALE to create a website that is does not provide the user with information. For instance, if you want to create an adaptive game, then GALE is certainly not the best option to go for. A truly generic adaptation engine that really supports all type of hypermedia does not exist at this moment.

One might ask what 'true adaptiveness' is anyway. In my opinion this has been achieved when the user is always provided with the information the user wants to see. This differs often from the information that the adaptation model thinks the users wants to see. Nowadays even the biggest online companies in the world have still not been able to determine what the user really needs. You can see this often when you have bought some shoes or booked a vacation online. Instead of showing the user real customised advertisements showing matching jeans or camping equipment, the user still gets stuck with advertisements about shoes or vacations. We are so close, yet so far away.

Chapter 5

Conclusion

As explained in Chapter 3, the GAM Authoring Tool has overcome a lot of problems previous authoring tools in existence had. Each time we had to take a choice under consideration, we have made a decision based on the three guidelines in Chapter 1. Next to this, all alternatives of each choices have been carefully weighed against each other. Hence we can conclude that the goal of this project—creating a new authoring tool for GALE that overcomes most of the problems previous authoring tools had—has been achieved.

However, there is a small point of improvement that we would suggest to people who want to improve the GAM Authoring Tool. As mentioned in Chapter 4 there has been no empirical research on the tool at all, due to the limited amount of time available. Therefore it would be a good idea to first run some test on the GAM Authoring Tool before continuing working on it and even during the development to ensure that the tool does meet all requirements.

Nevertheless, the implementation of all the improvements far outweigh the small possible improvements that could still be made. The quite extensive list of possible improvements in ALAT, which can be found in Section 3.2, has been reduced to nothing, as all these problems have been resolved and we are therefore quite happy with the produced result. Furthermore, we would suggest to continue working on the GAM Authoring Tool instead of starting from scratch again, because we think the principles of the tool are a good basis for everyone who wants to continue working on the tool.

Acknowledgements

Firstly I would like to express my sincere gratitude towards prof. dr. Paul De Bra and dr. ir. Natasha Stash who both have served as graduation supervisor and provided not only valuable and appreciated feedback during our weekly meetings, but also came up with several interesting ideas and alternatives that I could consider implementing in the GAM Authoring Tool, which has led to an even better result. Next I would like to thank dr. Kees Huizing for being part of the assessment committee. And last but not least I want to thank my family—especially Djamon—for their support and help throughout my graduation phase.

Bibliography

[1] W. Boereboom. ALAT: A new authoring environment for GALE. Master's thesis, Eindhoven University of Technology, 2016. 1, 6, 33, 37

[2] Peter Brusilovsky, Elmar Schwarz, and Gerhard Weber. A Tool for Developing Adaptive Electronic Textbooks on WWW. Published online: `http://aace.virginia.edu/aace/conf/webnet.html`, 1996. Accessible via: `http://www.contrib.andrew.cmu.edu/~plb/WebNet96.html`. 37

[3] P.M.E. De Bra, G.J.P.M. Houben, and H. Wu. AHAM: A dexter-based reference model for adaptive hypermedia. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany, February 1999. 31, 38

[4] P.M.E. De Bra, E. Knutov, D. Smits, N. Stash, and V.F.C. Ramos. GALE: a generic open source extensible adaptation engine. *New Review of Hypermedia and Multimedia*, 19(2):182–212, 2013. 1, 29, 37, 38

[5] P.M.E. De Bra, M. Pechenizkiy, K.A.M. van der Sluijs, and D. Smits. GRAPPLE: integrating adaptive learning into learning management systems. In *Proceedings ED-MEDIA 2008*, pages 5183–5188, Vienna, Austria, 2008. 3, 31, 38

[6] P.M.E. De Bra, D. Smits, and N. Stash. The design of AHA! In *HYPERTEXT '06 Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 171–195, Odense, Denmark, August 2006. 6, 31, 38

[7] P.M.E. De Bra, D. Smits, K.A.M. van der Sluijs, A.I. Cristea, and M. Hendrix. GRAPPLE: Personalization and adaptation in learning management systems. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010*, pages 3029–3038, 2010. 31, 38

[8] P.M.E. De Bra, N. Stash, Wouter Boereboom, Celine Chen, Joris Den Ouden, Martijn Kunstman, John Oostrum, and Egon Verbakel. ALAT: Finally an Easy To Use Adaption Authoring Tool. In *HT'16 : proceedings of the 2016 ACM Hypertext and Social Media*, pages 213–218, Halifax, Canada, July 2016. 33

[9] Number of Internet Users. Published online: `http://www.internetlivestats.com/internet-users/`, 2016. 1

[10] D. Smits. *Towards a generic distributed adaptive hypermedia environment*. PhD thesis, Eindhoven University of Technology, April 2012. 3

[11] D. Smits and P.M.E. De Bra. GALE: A Highly Extensible Adaptive Hypermedia Engine. In *HT 2011 - Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia*, pages 63–72, New York, United States, January 2011. 3

[12] H. Wu. *A Reference Architecture for Adaptive Hypermedia Applications*. PhD thesis, Eindhoven University of Technology, November 2002. 16, 21

Appendix A

GALE's Other Authoring Tool

A.1 Introduction

Over the last few years hypermedia have become more and more popular. While the number of users of hypermedia keeps increasing, not all users use a specific hyperdocument with the same goal in mind. In order to serve as much users as possible these hypermedia need to be adaptive, which means that the content and behaviour of a hyperdocument differs for each user based on the knowledge of the user or on some other criteria.

A.1.1 Introduction to GALE

GALE[4], a general purpose adaptive hypermedia engine, provides the functionality to create such adaptive hypermedia. Each project that is created in GALE consists of several *concepts* and *relationships*, which together form a Domain Model (DM).[4] Figure A.1 shows a fragment of such a domain model, that consists of the concepts 'Star', 'Sun' and 'Planet'. The arrows between these concepts represent their relationships, which tell you that a planet rotates around a star and that the sun an example is of a star.

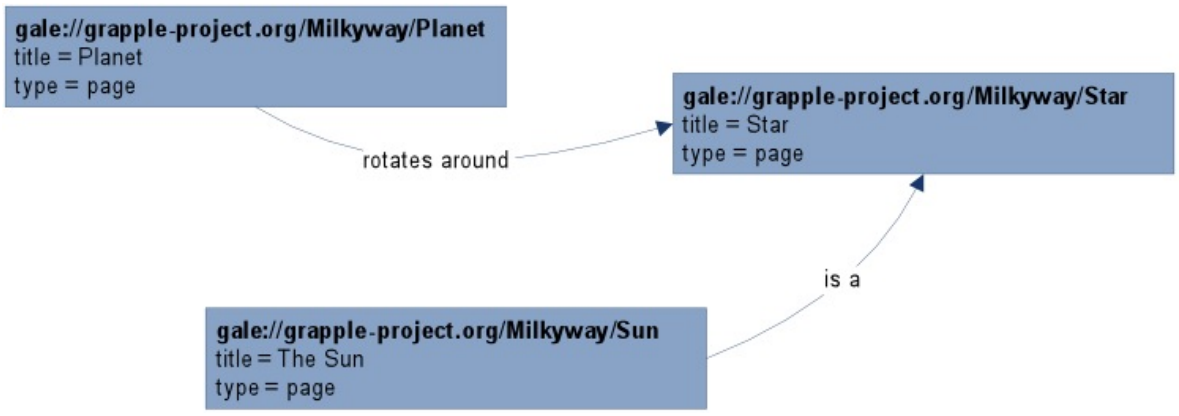

Figure A.1: A small fragment of the Domain Model.

For each of these concepts GALE keeps track of the user's interaction with that concept, by keeping information in so called *User Model* attributes. Next to this, GALE associates behaviour and adaption rules with each concept. These rules, in combination with different User Models, lead to a different appearance of a concept for each user.

Although it relatively easy for computer scientists to use GALE, it proofs to be rather difficult to use for people without any programming experience. In order to make this easier for those people, which still would want to make adaptive hypermedia, several authoring tools have been developed to achieve this goal, which I have already discussed in the literature study (see Appendix B) for this seminar.

A.1.2 Goal for my master's project

During my master's project I want to develop a new authoring tool for GALE that not only overcomes the problems that previous authoring tools for adaptive hypermedia had, but also overcomes problems that authoring tools in general have. Next to this, I want the tool to have some new functionalities as well. The specific problems that I want to overcome and the new functionalities are listed below:

- The authoring tool should be easy to use without any technical knowledge or programming experience.
- During each step the manual entering of code should be optional, which means that it should be possible for a user to write code manually, but there is always a way of achieving the same thing without doing writing code.
- The authoring tool has to be a general purpose tool, so it should not restrict to only one type of adaptive hypermedia applications, like electronic textbooks.
- The tool should be able to generate .gam-files—which are files used by GALE—from the structure that is created with the tool and vice versa.
- The tool has to be integrated with GALE, which means that no special installations are required to use the authoring tool.
- Creating concepts that follow *more or less* the same structure should be easy to do without doing anything twice. This can be achieved with the use of for instance blueprints.
- The new tool has to behave like a user expects it to behave.
- The visualisation of the concept's hierarchy should immediately be clear, without restricting the concepts to have only one parent.

A.1.3 Gaol for my seminar project

It is this last gaol for my master's project that my graduation supervisor recommended me to solve as a seminar project. I will do this by comparing several authoring tools and other

tools containing hierarchical structures. At the end of this paper you can read how I want to visualise the concepts in this new authoring tool for GALE and what the planning is for my master's project.

A.2 Comparison study

In order achieve the goal for the gaol for this seminar project, a more specific explanation about what this hierarchy does is needed. Assume you want to visualise all the concepts and relationships between them like is done in Figure A.1. You can imagine that this DM will become larger each time a concept is added to it, which will lead to an unclear, cluttered representation of the Domain Model. In order to overcome these problems a hierarchy is needed that structures these concepts in such a proper way.

A.2.1 Authoring tools over the years

Let us first take a look at previous authoring tools that have been developed in association with the Eindhoven University of Technology and that have been around in the last decade.

AHA!

In 2006 partial implementation of AHAM[3], which is a reference model to support support adaptive hypermedia authoring, emerged. This implementation, called AHA![6], came with a set of authoring tools, one of which is called the *Graph Author* tool. A print screen of this tool can be seen in Figure A.2. As you can see this tool contains of two parts: the left side of the tool shows a certain hierarchy of concepts that is determined by the person who used the tool, while in the middle the relationships between these concepts are shown, which are directly drawn from a template.

It is not difficult to see that this authoring tool has the same problem as Figure A.1: the graph will become more difficult to read as the number of concepts and relationships grows. The hierarchy on the left side stays however more or less readable, since the concepts are divided among several folders and you can hide and show the contents of a certain folder by clicking on it.

GAT

GAT is the authoring tool of GRAPPLE.[5] The GRAPPLE project aims at delivering a technology-enhanced learning environment to learners that guides them trough a life-long learning experience, automatically adapting to personal preferences, prior knowledge, skills and competences, learning goals and the personal or social context in which this learning takes place.[7]

It is intended to support learning based on the concept of courses and creating these courses is done by using several tools. One of these tools is the Domain Tool, which can be seen in

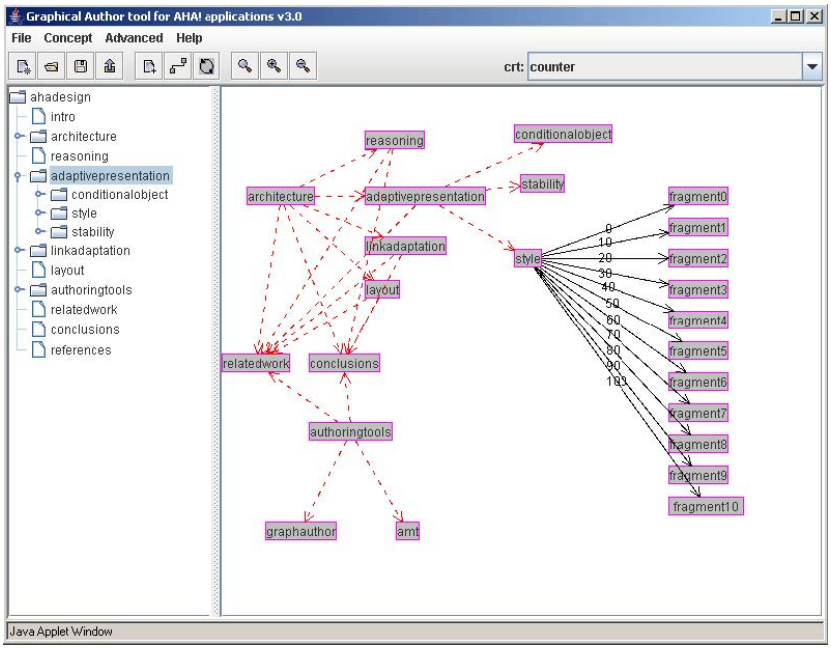

Figure A.2: The Graph Author for AHA!

Figure A.3. Just like AHA! this tool allows for binary relationships between concepts. The difference with AHA! is that these relationships are labelled, so it will become a richly interconnected collection of concepts.

However, once again it will be hard to keep the overview when the number of concepts and relationships grows. The zoom-functionality can be handy in these situations, but when the related concept is shown far away in the graph, this would not work. Furthermore, the hierarchy that you could create in AHA!'s Graph Author tool and allowed for quickly finding the right concept, is not implemented in this tool, which makes it even harder to find the correct concept.

Figure A.3: The Domain Tool in GAT

ALAT

ALAT[1] is the most recent authoring tool for GALE, which should have overcome all the flaws and shortcomings of the other tools. Since you will find quite an extensive list of possible improvements in my literature analysis (see Appendix B) and this chapter aims at giving different views on how to structure concepts hierarchically, we will only discuss the hierarchy of ALAT in this paper.

Figure A.4: Hierarchy in ALAT

This hierarchy can be seen in Figure A.4. The first thing that is noticeable in comparison with GAT and AHA! is that ALAT dropped the graph-like presentation of the concepts and their relationships, as such graphs almost always lead to unreadable structures when the number of concepts and relationships grows.

Instead, ALAT uses a hierarchy that was shown at the left side of Graph Author tool of AHA!. One big difference however, is that ALAT does not use folders to structure the concepts, but parent-relationships between concepts. As you can see the root concept 'Milyway' is the parent concept of the concepts 'Planet', 'Nebula', 'Star' and 'Moon'. You can view or hide the children of each parent by clicking on the arrow at the left side of the parent concept.

A.2.2 Other tools

Since another paper[8] on ALAT already discusses several other authoring tools that are related to adaptive hypermedia and came to the conclusion that a graph-like visualisation of the concept hierarchy almost always leads to unclear situations when you have multiple concepts and relationships, two other tools that are not related to adaptive hypermedia are discussed in this section. Both tools are widely used and have a solution for visualising hierarchical structures.

File explorer

The first tool that deals with hierarchical data and is widely known among most people is File Explorer, also known as Windows Explorer. This program provides the users of a computer running a Windows operating system with a graphical user interface that can be used to browse files. In Figure A.5 you will find a print screen of this program. With this program you can browse trough the files on your Windows computer. Each file is stored in a folder and the structure of the folders and so called 'favourite' folders can be seen in the left. Once you click on a folder, the contents of this folder is shown in the middle, which can be any file on your disk or another folder.

If you would translate this to a way to store concepts in GALE there are several things to take into account. First of all it becomes hard to determine what the 'main' concept of your application is, i.e. the concept that is first shown when someone visits the application. Secondly, although you can view the folders that are contained by several folders at the same time, by using the left side of the program, there is no fast way to view all the contents of several folders at the same time. Lastly, this model assumes that a concept can only be stored in one folder, while the new authoring tool should allow a concept to be stored at more than one place in the hierarchy.

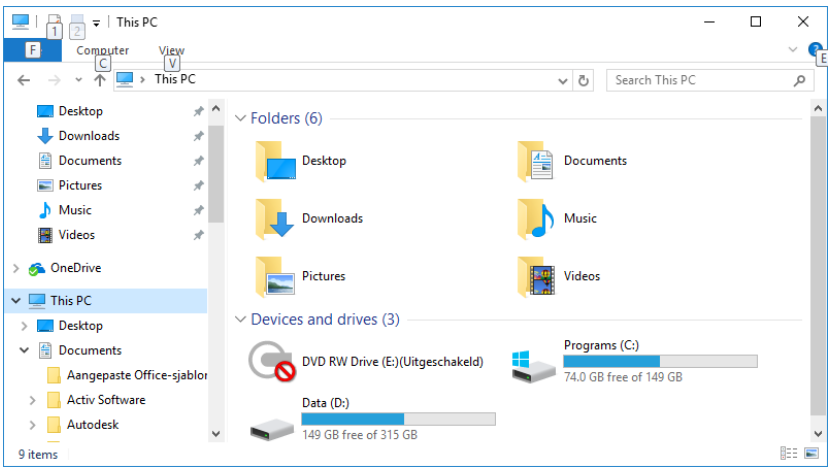

Figure A.5: A print screen of File Explorer

Adobe Photoshop

The second tool I want to discuss is Photoshop, which provides the user with the functionality to structure the layers of each project hierarchically, as you can see in Figure A.6. Just like File explorer, each layer is stored into a certain group. One great feature about this grouping is that you can assign certain properties to the whole group at the same time, which could never be done in one of the authoring tools that were discussed in this paper. These same properties for the whole group also apply at the visibility of each layer in Photoshop, which can be a useful functionality in an authoring tool for adaptive hypermedia as well. If you would allow the user to remove a (group of) concept(s) from the website by only one click without

actually deleting the concept(s), you can start adding concepts and relationships long before you actually want them to be on your web page.

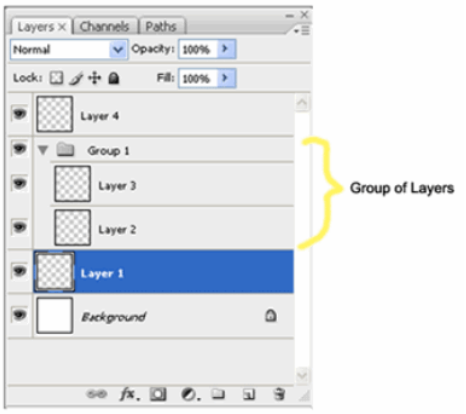

Figure A.6: Grouping layers in Photoshop

A.3 Results and work to do

A.3.1 Results

I think the best way of structuring the concepts in the new authoring tool is based on combining the powers of ALAT's structure with the structure that Photoshop uses. What I mean by this is that I want to have two ways of grouping concepts. The first is to use a parent-relationship between concepts in order to place certain concepts *under* their parent concept, just like ALAT does at this moment. Sometimes however, it is preferred not to have a parent-concept, but just a parent-folder. In this way you can categorise concepts that should have similar properties, but without having a special parent-concept.

A good example is classifying the food in your kitchen. Assume you want to make a course on the different food that is eaten in the world. It would make sense to make several parent-concepts based on the origin of the food, like Asian food, Italian food, et cetera. The parent-concepts contain other concepts like noodles (for Asian food) or pasta (for Italian food). Both are made of unleavened dough though, so it would be a good idea to store them in a folder containing all food made of unleavened dough. Just as with the parent-concepts, you can define shared properties for all concepts in the same folder. However, the users cannot see that concepts are stored in the same folder, nor is there any page—on unleavened dough in this case—describing the folder.

With one simple click it should be possible to hide and show a (group of) concept(s). We have to keep in mind that the main-concept is always on top of the hierarchical tree and that each concept can be stored in multiple places. Internally there is only one copy of each concept, which shows up at multiple places in the hierarchy. There is only one restriction on storing the concept in multiple places, which is that it is not allowed to have a loop in the hierarchy. This means that a concept cannot have itself as a(n) (in)direct parent.

In short: the new authoring tool will use a layout that is more or less the same as ALAT, except (i) that we can store the same concept at multiple places in the hierarchy, (ii) that instead of using a parent-concept, concepts with similar properties can also be stored in a folder and (iii) there is an eye on which you can click to quickly publish or unpublish a concept.

A.3.2 Planning for my master's project

The planning for my master's project is quite complicated and should be well considered before starting on it, due to several inconveniences. First of all my graduation supervisor, Paul De Bra, plans to retire in October 2017. Although it is possible to be assigned to another graduation supervisor, I prefer to have the same supervisor for whole project.

Next to this I am following the Computer Science master at the same time as the master Science Education and Communication, which means that during the school year I have to teach computer science at the *Sint-Janslyceum*, which is a secondary school in 's-Hertogenbosch. Since the usual planning for a master project is working full-time on the project for six months, I have to do things differently, due to my partly unavailability.

This means that I want to work on my master project half-time for four months (February until May). After these four months I have another four months left (June until September) in which I want to work full-time on my graduation project. If all goes according to plan, which I feel is possible, I have one additional month left, which gives me a small buffer if something would go wrong.

During the first three months I want to do as much research as possible, because some of the goals described in Section A.1.2 can go wrong if not thought about them carefully. For example, I have to study how ALAT works, how it stores the data and how it generates the .gam-files for GALE. Since I want the 'generate' button to work in two ways instead of one, I have to think of way to do this, because it is possible for two different ALAT projects to produce the same .gam-files, but when I want to read the .gam-file again it has to produce the correct project for the authoring tool. This can probably be solved by adding comments in the .gam-files which can be read by the authoring tool.

This kind of study will be done in the first three (half-time) months. Once I start writing the actual tool, I plan to work two full-time months on it. This leaves me with one and a half month left, which can be used for testing, improving and finalising my final report.

Appendix B

Literature study

B.1 Introduction

During my research project I want to do something in the field of adaptation. Although I am not sure what the specific research questions will be, I want to develop an alternative for ALAT[1], which is an authoring tool for GALE[4] developed by a master student in 2016. He stated that the major challenge in the creation of adaptive hypermedia application lies in its complexity, and wanted to make sure that non-experts or novice GALE developers could easily make an adaptive web application, since the authoring tool at that time did not contain the expected and required functionalities for that. In order to overcome these problems, he developed the authoring tool ALAT.

In this literature survey I will look at some of the previous authoring tools for GALE and its predecessors that were available before ALAT was used and highlight for each tool the pros and cons of that tool. Next to that, ALAT has some shortcomings too, which I want to overcome with the new authoring tool. At the end of this literature survey I want to be able to specify a clear goal about the requirements of the new authoring tool I am going to develop.

B.2 Authoring tools for hypermedia in existence

In this section I will summarise for some authoring tools for GALE and its predecessors the pros and cons of the tool. As this is already done by the student who created ALAT[1], I will summarise his findings and focus on the strengths and shortcomings of ALAT, since I want to develop a successor of ALAT.

B.2.1 Interbook

In 1996 a description of an authoring tool called Interbook[2] was published. Using this authoring tool it was possible to turn a normal textbook into an adaptive electronic textbook. This is done by starting with an hierarchically structured file in Microsoft Word. By annotating the Word file and exporting it as HTML you can create an adaptive electronic textbook that

fits your needs. At that they recommended you to use Netscape as a browser, due to the 'advanced' features like frames and several windows. Nowadays almost all browsers support these frames, although some are dropping the functionality again, as these tags are not included in the new HTML 5 standards.

By annotating the original textbook in Word, it is fairly easy to create an adaptive website, since most people are familiar with this program. Furthermore it is easy learn how to annotate those documents, so you don't really need technical knowledge to make something adaptive.

The drawback however is that you have to manage all the concepts and rules manually and there is no support that helps you creating the adaption models. Next to that, Interbook is designed to create adaptive electronic textbooks and as a result of that it is not easy to create modern adaptive hypermedia applications.

B.2.2 AHA!

Just like Interbook, most researchers have concentrated on a single application or application area for their adaptive hypermedia systems. The architecture of many of these adaptive hypermedia systems has been captured in the AHAM reference model[3], which resulted in a partial implementation of this model in 2006 called AHA![6], with which a new set of authoring tools emerged, one of which is called the *Graph author* tool.

With this tool the user doesn't have to deal directly with that adaption model code that AHA! uses, which makes it easy for non-technical users to use. It is great at creating and showing relations between the different concepts. The drawback of this is that in case of multiple relations, the Graph Author becomes difficult to read, which results in losing the overview. Furthermore it is not possible to declare additional attributes for the concepts that do not match the concept templates.

B.2.3 GAT

GAT is the authoring tool of GRAPPLE[5]. The GRAPPLE project aims at delivering to learners a technology-enhanced learning environment that guides them through a life-long learning experience, automatically adapting to personal preferences, prior knowledge, skills and competences, learning goals and the personal or social context in which the learning takes place.[7]

It is intended to support learning based on the concept of courses and creating these courses consists of three steps in the GAT. At first you determine the concept of each course in the Domain Model Tool. At second you have to add relationships between these concepts in the Course Adaption Tool, which are used to determine which courses are recommended or not. At last you have to write content for these concepts, which is written in XHTML in order to be used with GALE, which is developed as a stand-alone adaptive learning environment in the GRAPPLE project.[4]

The implementation of these several steps has a couple of shortcoming. First of all you loose track of the relation between the domain and adaption model. And once you have to make rules for concept, you have to match concepts to rules instead of the other way around, which is counter-intuitive.

B.2.4 ALAT

ALAT is the most recent authoring tool for GALE, which should have overcome all the flaws and shortcomings of the other tools. But as always this leads to new ideas for improvements. ALAT is an easy to use tool, in which you can create project. In each project you can add several concepts that match a certain blueprint.

The first drawback that comes to my mind when using the tool is that is not possible to add, edit or delete the blueprints that are used to create the concepts. In order to do that, you have to edit a JSON-file manually, which seems like a difficult job for non-technical people.

Furthermore it is not possible to 'drag' a concept from one place to another. If you accidentally added a concept in the wrong place, you have to delete that concept and create it again in a different place in the hierarchy. Even the assumption that each concept can only have one parent restricts the tool to uses that fit this hierarchy.

Next to this, clicking the mouse doesn't result in the behaviour you expect. After logging in in ALAT you have for instance choose a project you want to work in. Instead of just double clicking on that project, you have to click the project once and click on a separate button that says 'load'. These kind of irritating missing features result a tool that is—at least for me—annoying to use for people who expect fast interaction.

But most important of all is that ALAT is developed as a stand-alone website that can create files for GALE based on structure you have created with the tool. Each time you make a small change, you have to generate these files again. The ultimate solution for this would be a need integration with GALE, so you can edit the application either using the tool or by editing the code manually. Once you've edited the application manually, the tool should detect these changes and adapt itself to the new situation.

There are many more points to improve, but these will hopefully become clear once I have studied ALAT more thorough.

B.3 Goal for the new authoring tool

I divided all the pros and cons from the tools described in the previous section into two categories. The first category is 'requirements' and contains the specifications the new authoring tool has to meet. The second one, called 'wishes', contains specifications that I would really like to process in the new authoring tool, but have a lower priority than the requirements.

B.3.1 Requirements

- The tool should be easy to use without technical knowledge.
- Avoid that the user has to enter code anywhere, but if he wants, it is still possible.
- I shouldn't restrict to one type of adaptive hypermedia applications, like electronic textbooks.
- I only want to have *one* tool that does it all, instead of developing multiple tools.

- The tool should use blueprints, which are easy to create and modify by non-technical users.
- The tool should be able to generate GALE files from the structure that is created with the tool and vice versa.

B.3.2 Wishes

- When the tool visualises a big set of relations, it should make sure that everything remains readable.
- It should be possible for a concept to have multiple parents.
- Don't use buttons for everything, but make sure that double clicking with you mouse results in the expected behaviour.

In short: There are still many improvements to make in the area of authoring tools for adaptive hypermedia. During my project I want to develop a new authoring tool for GALE that is seamlessly integrated with GALE itself and overcomes the weaknesses of authoring tools is the past.