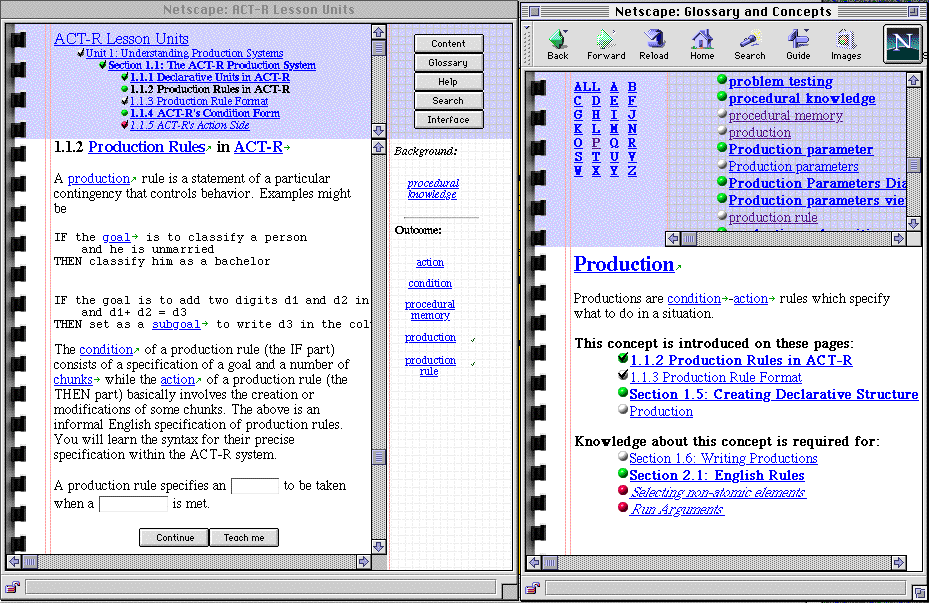
# 2. The past and present

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 Previous authoring tools

When looking back at the beginning of adaptive hypermedia research, a first extensive description of a published authoring tool was on Interbook(1) back in 1996. Interbook is a system used in the authoring and deployment of adaptive electronic text books. An example of an Interbook application can be seen in *figure 3.*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 by the current concept. Both of these types of adaptation rules are defined by the annotations made during the authoring of this electronic textbook.

Figure 3. 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 further lowers the barrier of entry and helps to get the author started quickly.  
This first form of authoring is easy to learn and has a familiar environment to author in. Yet, it does have some drawbacks that make it difficult to use in more modern authoring systems.

First of all there is no 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 their own at tackling these issues to create a more generic and extendable platform.

A new set of authoring tools emerged with the development of AHA!(2). 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 [REF PREVIOUS SECTION]. It is also a direct predecessor of GALE, for which the authoring tools will be treated later this section. The main authoring tools developed for AHA! are the Concept editor and Graph author[[1]](#footnote-1). 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 has 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.

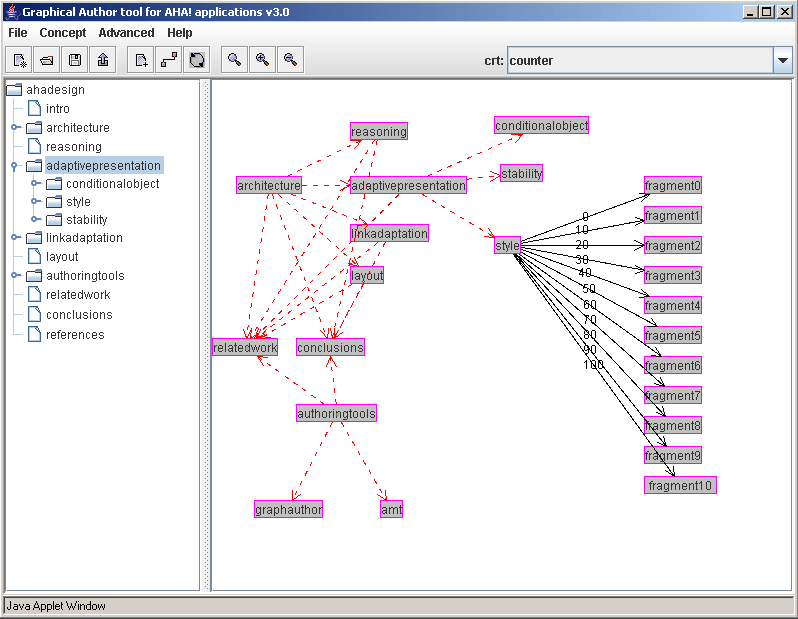
When looking at the graph author interface in *figure 3.2*, there is an indented tree structure which shows the hierarchical topic structure in the shape of parent-child relationships on one side and a graphic representation of the course and its relations on the other. The hierarchical structure is 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 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.

Figure 3. 2: The AHA! Graph Author

However, it can already be seen in [FIGURE REF] that the graph interface which holds all pedagogical rules and additional relations becomes cluttered quite quickly, making it easy to lose sight of 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 .

## 2.2 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 section [SECTION REF].

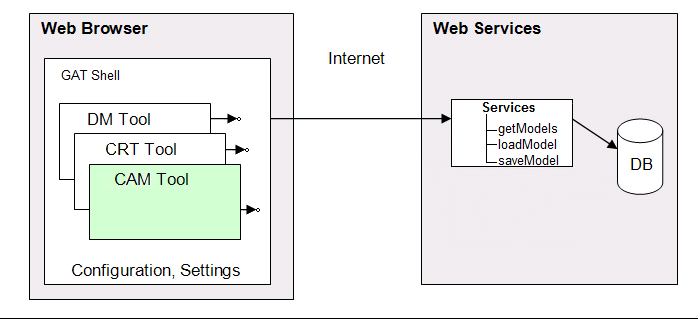
This has resulted in distribution of tasks in which multiple parties created different aspects of GAT. The separation of tasks amongst different parties has caused further separation between different aspects of content authoring. The authoring of domain and adaptation model have separated to a point at which they are created in separate tools. There’s 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 3.3*. The DM Tool represents the Domain designer, the CRT tool 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.

Figure 3. 3: The GAT Architecture

### 2.2.1 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 lets the user 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 a 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.

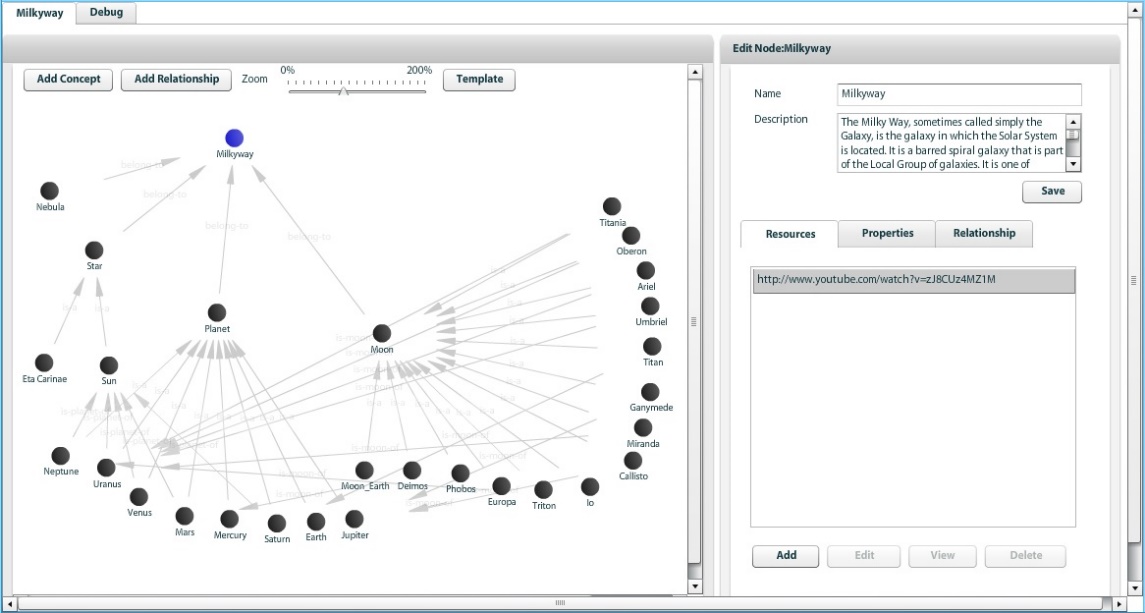
The models created in this tool have a graph structure in which the user can add or delete nodes and relations. Unlike the nodes created in the graph author, each node and relation in these editors also has their own set of properties which the user can edit.

Figure 3. 4: The GAT Domain Designer

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 3.4*, the interface becomes cluttered as the domain model grows.

Adding a new relation to the domain model involves a separate menu with select boxes which contains 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.

### 2.2.2 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.

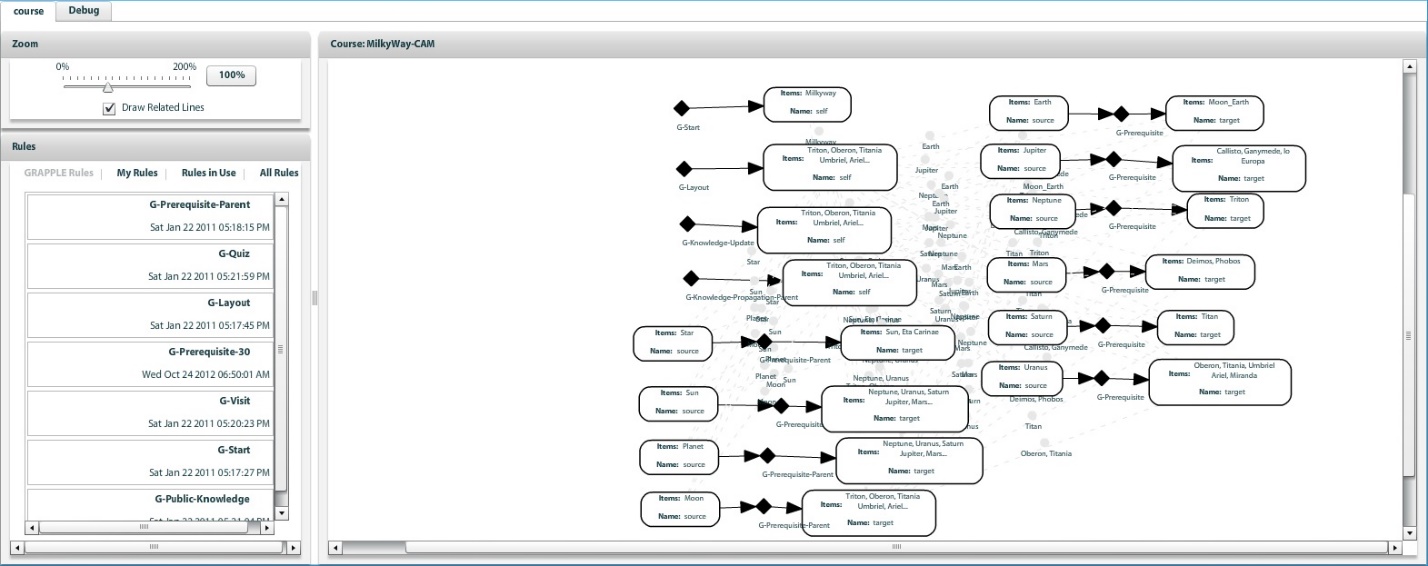
The course designer also uses the same drag and drop graph-like visual interface as the domain designer. It takes domain concepts and introduces rules and constraints on them such as prerequisite constraints, knowledge propagation rules or starting topic rules. GAT provides a predefined set of rules, created using the pedagogical relation (PRT) described below. To create custom rules and constraints one must use PRT. After restarting the course designer these new rules then 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 3.5*).

Figure 3. 5: The Gat Course Designer

During the GRAPPLE project, a “standard” set of 15 rules were 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 section regarding setting up new relations in the domain designer. This makes it hard to easily choose the right 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).

### 2.2.3 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 3.6)*.

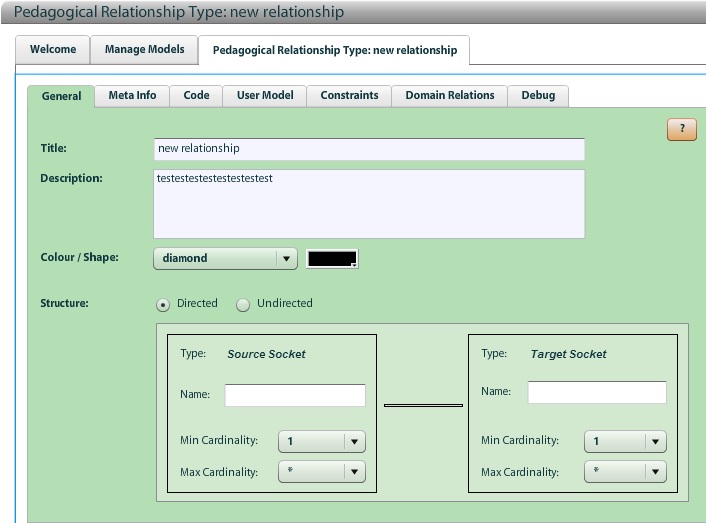


Figure 3. 5: The Gat PRT

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.

### 2.2.4 A toolset divided

The decision to separate the authoring process into multiple tools have made it difficult for new users to understand the connection between 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 REFERENCE], the two workflows can be molded back into a single, more intuitive workflow.

So when regarding the GAT-workflow, we can conclude that the separation of the adaptive course authoring process into a domain and course model make 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 barrier-of-entry. 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.

### 2.2.5 Techniques

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

First off 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 3.7)*.

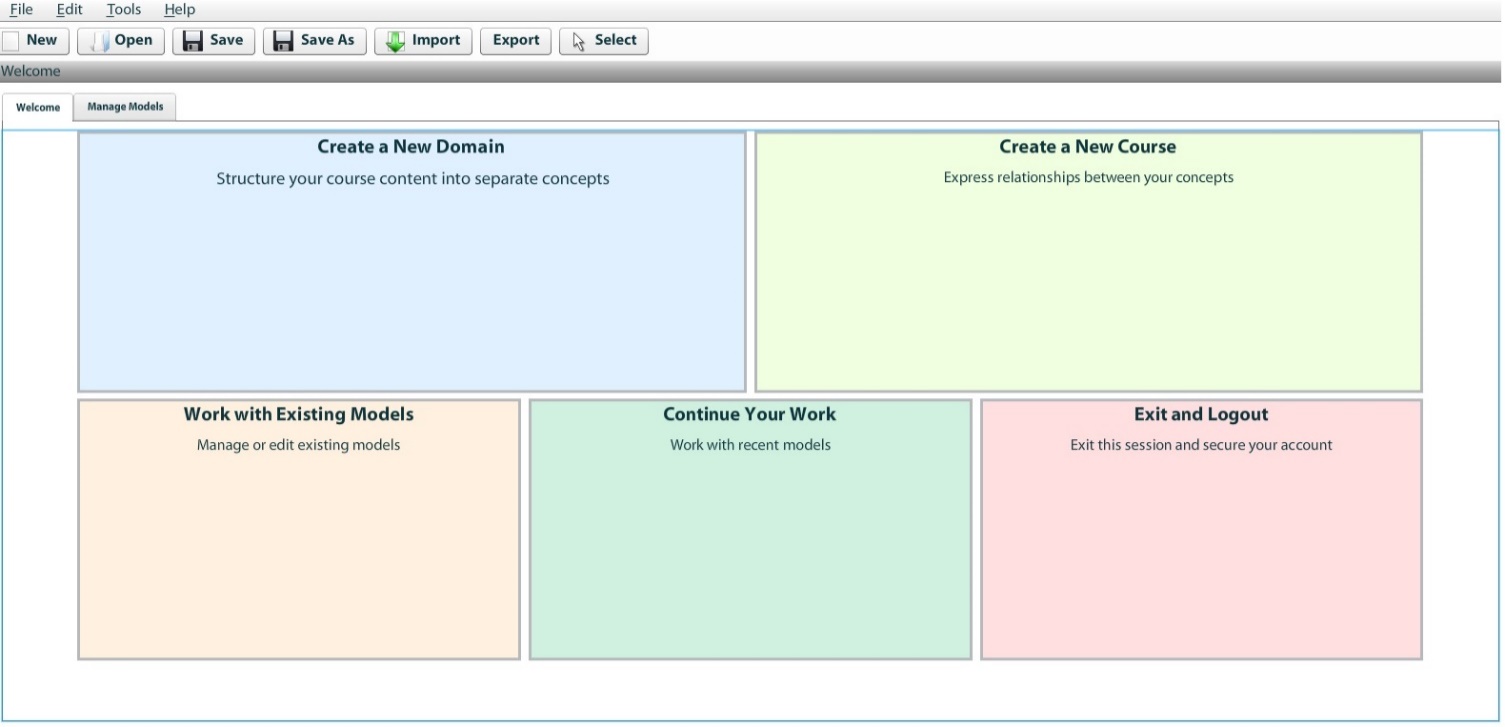


Figure 3. : The GAT Welcome Interface

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.

## 2.3 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 section. It should have a user friendly non-cluttered interface and workflow, allowing the authoring of advanced adaptive applications with ease and a low barrier of entry.

**References:**

1. 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.
2. De Bra, Paul, David Smits, and Natalia Stash. "The design of AHA!."*Proceedings of the seventeenth conference on Hypertext and hypermedia*. ACM, 2006.
3. David Smits. “*Towards a Generic Distributed Adaptive Hypermedia Environment.*” (http://gale.win.tue.nl/thesis)

1. AHA! Also has a Form editor but we will not discuss this special-puspose tool. [↑](#footnote-ref-1)