

# Authoring Content Structure for Adaptive Documents

Shijian Lu and Cécile Paris

CSIRO ICT Centre, Locked Bag 17, North Ryde, NSW 1670, Sydney, Australia  
{Shijian.lu, [cecile.paris@csiro.au](mailto:cecile.paris@csiro.au)}

**Abstract.** As information overload is an increasing problem, the benefit of providing appropriate information to satisfy users' special information needs is widely recognised. However, developing adaptive systems is not a trivial task. One of the challenges is in specifying the types of text that are expected, under which circumstances. In this paper, we present Constructor, a graphical authoring tool to be used in conjunction with an adaptive system based on linguistic principles. Constructor allows for the authoring of document content and structure from which the resources required by the adaptive engine can be generated automatically.

## 1 Introduction

As information overload is an increasing problem (e.g., [8]), the benefit of providing appropriate information to satisfy users' special information needs, whether they be learning, entertaining, or working information needs, is widely recognised. These are largely driven by the context in which they occur. How to exploit contextual information to deliver what users need is the central theme of adaptive systems research. Studies have shown that documents containing information adapted to the needs of individual users outperform general purpose documents (e.g., [3]).

Given the effectiveness of adaptive information, the question of how to develop adaptive information systems is an important one. To address this question, different approaches have been explored. One approach is underpinned by pedagogical principles, e.g., [7]. Here, adaptation is chiefly based on matching users (learners) *existing* knowledge with *prerequisite* knowledge as specified in a closed set of learning material. In this approach, documents are manually authored, together with specifications of the circumstances under which the text should be presented, and a user model is implemented as an overlay of an underlying concept model. Each concept is associated with a value indicating a user's current state of knowledge with respect to that concept. An adaptive engine can compare the value of a concept in the user model with the conditions of a concept in its conceptual model in order to filter out a sub-hypermedia space together with the corresponding navigation support to suit a user's goal and knowledge level.

In another approach, the linguistic notions of *relevance* and *coherence* are exploited to produce the appropriate tailored information space [14]. This approach is

centred on a generation engine, libraries of operators, a set of contextual models and a set of retrieval operators that serve as the interface with the data sources. The set of contextual models includes the users' characteristics, their task at hand, and the device they are currently using or the environment in which they are and the dialogue history so far. These contextual models can be used to constrain how to select, organise and realise the information. Here, adaptation is chiefly based on the constraints that are provided by the contextual models on what information is to be presented to the user and how to present it. In this approach, unlike in the former one, systems typically work with an open set of (potentially heterogeneous) resources, and the final text/presentation is generated automatically.

Both approaches have evolved along a similar pattern in terms of research, namely, (1) *special purpose* system stage, (2) *general purpose* system stage, and (3) *authoring tool* stage. In the *special purpose* system stage, a system is built for a specific application. In the *general purpose* system stage, focus is shifted towards developing software environments or platforms to facilitate the creation of different types of adaptive applications (e.g., [6] and [14]). Finally, in the *authoring tool* stage, researchers turn their focus on developing authoring tools such that, instead of relying on technological experts, application domain experts can be empowered to develop new applications. For systems working with a closed set of resources, there are already some authoring tools (e.g., [2] [5]). In this paper, we present an authoring tool for a system working with an open set of resources. The tool is not only to augment an existing set of resources (as in [1]), but it also allows one to define new texts that an adaptive system is to generate, and it can be used in a variety of applications.

The paper is organised as follows. We first present a brief overview of our platform for developing adaptive systems, Myriad and the VDP, which is based on linguistic principles, and we discuss the need for an authoring tool. We then present Constructor, a graphical tool for authoring new documents. The usage of the Constructor is illustrated with a simple application. The paper concludes in section 4.

## **2 An Adaptive System with an Open Set of Data Sources**

### **2.1 Myriad and the Virtual Document Planner (VDP)**

Myriad is our platform for Contextualised Information Retrieval and Delivery [14]. Its main module is the Virtual Document Planner (VDP), a multimedia generation system that dynamically produces tailored hypermedia documents, integrating several heterogeneous data sources and customising the content for a user. The VDP is based on a typical Natural Language Generation (NLG) architecture, where the linguistic resources are separate from the engine. As in [13], the specific engine that the VDP employs is a planning engine, and the resources are represented as plan operators. The VDP (and Myriad) belongs to the class of AH systems working with an open set of (potentially heterogeneous) resources. Instead of providing navigation support and guidance through the (manually authored) hypermedia space, the VDP *creates* a tailored hypermedia space. The VDP generates hypermedia documents using

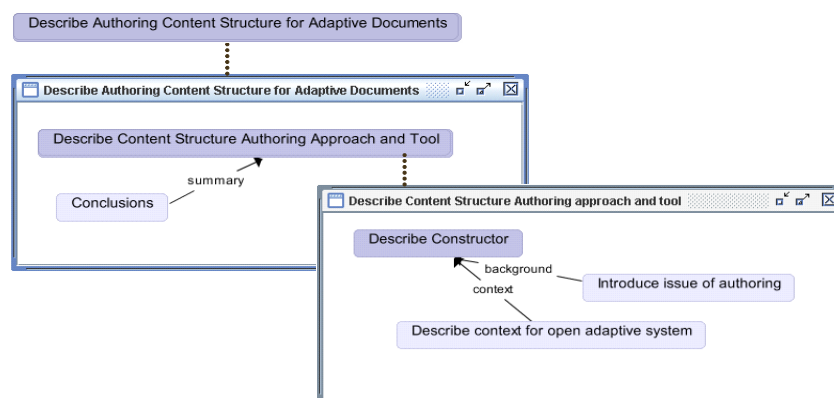
discourse operators to select and organise the content, presentation operators to determine an appropriate way to realise the content and the structure of the presentation, and a set of templates to control the layout of the final document. It is through the discourse operators that one specifies the types of text to be generated (content and structure). A discourse operator is essentially a plan that specifies how to achieve a communicative goal. It decomposes a goal into sub-goals, ensuring that the sub-goals are related by rhetorical relations, as this is what will ensure the coherence of the final text/presentation. Discourse operators typically have constraints on their applicability. The constraints usually refer to the contextual models (e.g., the user model), or the underlying data.

The development of an adaptive information system using this approach thus requires the authoring of the discourse operators. These are typically manually written. They are not easy to write, however, as they require expertise in the domain, writing skills, computational linguistic skills, and, of course, knowledge about the required specific syntax. Our aim in this work is to provide a high level authoring tool that a domain expert with writing skills can use to specify the types of documents to be generated and their applicability (e.g., for which users, which context, etc.).

## 2.2 Specifying document structure and content

We would like people knowledgeable about the texts required in their domain and with writing skills to be able to specify contents for the documents that an adaptive system is to produce. To this end, (1) we provided an abstraction on the discourse operators, allowing authors to specify the structure and content of a text without having to know much about computational linguistics nor about the specific syntax required for discourse operators, and (2) we decoupled the specification of the structure of a text from the knowledge of how to retrieve data and from the specification of the applicability of the structure (i.e., the specification of the constraints for an operator).

We introduced the concepts of *content structure* and *retrieval registry* [11]. The former is illustrated with respect to this document with the figure below (Figure 1).



**Figure 1.** The high level content structure for this article

At the highest level of abstraction, the purpose (communicative goal) of this document is to describe an approach to authoring the resources required for an open source adaptive hypertext system. There are 2 main parts, or *text spans* (as defined in [12]), to the article: one where the idea for the paper is introduced and expanded (corresponding to sections 1, 2 and 3), and another one which summarises the content already presented (corresponding to section 4). At the next level of abstraction, the first text span can itself be decomposed into 3 smaller text spans which, again, are not independent of each other and play different roles within the paper as a whole: one span describe the Constructor tool (Section 3), which is the main information for the article, while the remaining text spans provide supplementary information (some background, the issue at hand, and some context, a description of our adaptive system).

A content structure is thus an abstract document definition model, defined in terms of the purpose of the various parts of a document and their relationship. It is domain dependent and can be authored by someone who knows how to write in the application domain. The retrieval registry is a library of retrieval functions or specifications, called "*retrieval services*", which specify how to get the data. Given the content structure and the underlying retrieval registry, the authoring tool constructs the discourse operators required by the delivery engine.

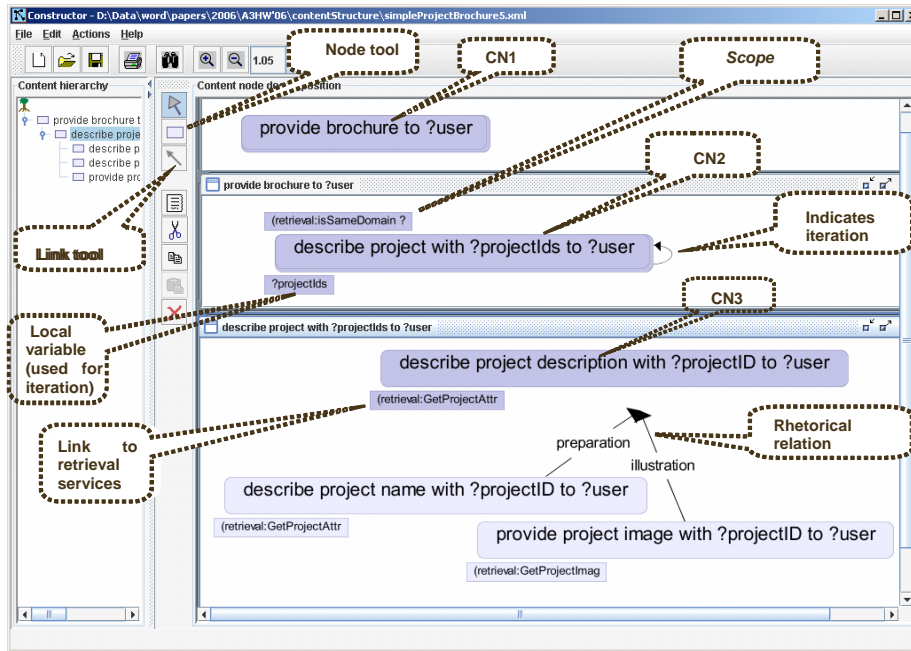
A content structure is composed of content nodes and relationships among them. Content nodes essentially correspond to the communicative goals defined in the discourse operators, or, looking at a resulting document, they correspond to text spans, or discourse segments, each presenting some "chunk" of information, with a purpose. Nodes at different levels of abstraction are governed by hierarchical relationship while sibling nodes are related with rhetorical relations (as defined in the Rhetorical Structure Theory [12]). A content node may have zero, one or many children nodes. It may have one to many parent nodes, since a sub-structure can be used in a number of documents. Content nodes thus define the structure of the documents to be produced, in terms of the respective communicative goals of the various text spans, also specifying how different discourse segments relate to each other (through the rhetorical relations), and the relevant scope for any particular node in relation to any contextual model. Finally, content nodes may link to retrieval services such that appropriate data can be retrieved from the underlying knowledge sources.

### **3 Constructor: A Content Structure Authoring Tool**

To facilitate the authoring of the content structure, we have developed a graphical authoring tool called Constructor. Our goal was to enable domain experts to develop content structures by either creating them from scratch or by reusing existing ones.

Figure 2 shows Constructor's main window. There are three regions. The left region is the content hierarchy pane, where all content nodes are displayed. This can be likened to a table-of-content (TOC), although it is not a TOC of the possible documents, but rather a conceptual view of the hierarchy being defined. (It is not a TOC because order amongst sibling nodes is not determined here. It is determined

during the generation phase based on the rhetorical relations and the context.) The middle region is the tool bar for creating or manipulating content structures. The right region is the node decomposition pane which is the main working area.



**Figure 2.** The Constructor's main window

To author content structures, authors need to think about document structure and relationships between different chunks of information, not about the phrasing proper of the document. Going back to our previous example about this article, as authors we had to think of what information we wanted to present and how the different parts fitted together, establishing a content structure like the one in Figure 1.

Authoring a document via Constructor and writing a document directly share some concerns but also have differences: while an author needs to think about document structure and relationships amongst different sections in both cases, this thinking must be explicit when using Constructor. Each content node must have a purpose (a communicative goal, or discourse purpose), and sibling nodes must be related with rhetorical relations. There are other differences between these two processes:

1. The result of writing a document produces a specific document (an instance of a document), while the result of authoring a content structure is a document definition model from which a variety of instances can be generated.
2. Although the granularity of information chunks may differ from one application to another, someone writing a document must know the actual content to include. In contrast, when authoring a content structure, an author only needs to point to an appropriate retrieval service to acquire the actual content. The exact content will then depend on when the retrieval service itself is called (allowing for dynamic data to be always accurate in the generated document).

3. Apart from creating the actual content, an author writing a document needs to explicitly order the content in some specific sequence and choose the specific vocabulary and syntax. In contrast, an author creating a content structure does not need to worry about realisation nor presentation order. This will be taken care of by the presentation strategies.
4. Because, when using Constructor, an author specifies a definition model from which a number of documents can be generated, the author needs to explicitly think about the applicability *scope* for a content node, namely under what condition the content node will be included in the document to be produced. This mechanism is one of the mechanisms that enable the resulting system to be adaptive: depending on the exact context, different documents will be generated.

### 3.1 Main Functionality of the Authoring Tool

The Constructor offers an extensive set of functionality to author content structures. The following is a summary of its main capabilities.

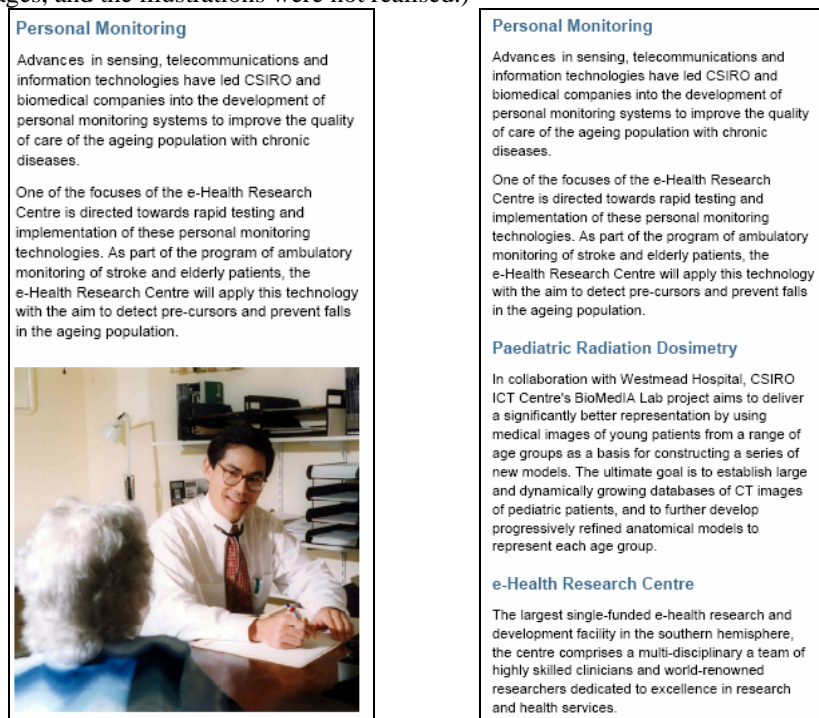
1. Basic functionality: This allows one to create, copy, cut, paste and delete content nodes and relations amongst them. The tool supports both top-down and bottom-up approaches to content structure authoring as well as content structure reuse.
2. Advanced functionality: This includes:
  - a. *Service parameter nesting* for the specification of parameters for the retrieval services. The parameters can call other retrieval services to allow more flexibility and power.
  - b. *Variable definition* for setting local variables, useful when one wants to refer to an object several times.
  - c. *Iteration definition* to specify that a content structure is to be instantiated several times, over a collection of items.
3. Discourse operator generation: The Constructor can generate discourse operators directly from content structures. This way, technical experts can later fine tune the generated operators if required. We envisage that it will be possible in the future to load the modified operators back into Constructor.

Given the space constraints, we cannot elaborate on all the capabilities listed above. Instead, we will provide one simple example of how to create a content structure from which different documents can be generated.

### 3.2 An Example

We illustrate how to use Constructor to create content structures using a specific application, Scifly [10], which delivers brochures about CSIRO's ICT research tailored to the user's interests. Scifly has been demonstrated in CEBIT Australia 2005/2006 [9]. SciFly can generate different types of brochures, all containing a variety of information, including an ICT centre overview, a domain description for the projects under consideration, details of the projects and contacts (scientific and business). For our example, we focus our attention on defining a content structure for

a (simplified) project brochure, such as the ones shown in Figure 3. The brochure in 3(a) corresponds to brochure about 1 single project (Personal Monitoring), while the brochure in 3(b) is about several projects (Personal Monitoring, Paediatric Radiation Dosimetry and the e-health Research Centre). (You may note that the brochure about the 3 projects does not contain pictures – this is because the presentation strategies and space constraints mechanisms in Myriad later reason about the instantiated discourse structure and the available space given the “device model” of the user to decide on what to realise in the end. In this case, the paper brochures are restricted to 2 pages, and the illustrations were not realised.)



(a) Simple project brochure with one project

(b) Simple project brochure with three projects

**Figure 3.** Generated simplified project brochures.

**Creating a Content Structure.** We now need to create a content structure to specify a project brochure. As authors, we thus need to think about what information we want to include in the brochure, and how we want it organised. In this case, we want the brochure to provide some information about each project selected, including its name, its description (from the underlying XML text database), and an image illustrating the project if there is one and there is space. This will result in the structure shown in Figure 2.

We now go through the process of creating this structure in more detail. Assuming we took a top-down approach, we first defined CN1, the content node corresponding to the overall brochure, giving it the communicative purpose: *provide project*

brochure to *?user* (the top sub-window in Figure 2). Here *?user* is a variable referring to the user model. It will be instantiated at runtime.

We now need to decompose that node. In particular, we want to specify that we need to find all the projects that the user was interested in and loop over them to provide a description of each project. This is done by (1) retrieving from the user model the list of projects the user specified in his or her query (not shown in the Figure), (2) binding this list to a local variable (*?projectIds*) so that we can refer to it, and (3) defining a new content node that will be iterated over once for each element of this local variable (so for each selected project). This corresponds to CN2 in Figure 2. We can see the iteration symbol, and the local variable that serves as the iteration variable. (What is not shown in the figure to avoid obstructing the figure is how this variable is set: it is set through a retrieval service available from the user service registry: (*user:getInterestedProjects ?user*). This service obtains the appropriate information from the user model.) This process illustrates the use of retrieval services to obtain data, providing service parameters to retrieval services (*?user* is a parameter to the service which will retrieve the projects the user is interested in – while it is a simple variable in this case, it could have been a (nested) call to another retrieval service), the use of a local variable and the iteration feature, which allows one to create only one content structure but specify that this structure is to be repeated for each object in the list.

We can now specify the structure for the node that will actually provide a project description for one project, CN3. It includes the main information (the project description, obtained through a retrieval service), a title (its name), and an image if there is one. Rhetorically, project name and image serves as supplementary information to the main information (the project description), related to this main information with the relation “preparation” and “illustration” respectively, as illustrated in the figure.

Figure 2 shows another feature of Constructor: the specification of the scope of a content node. This allows one to specify under which context the content structure is applicable. In the figure, CN2 has a scope, which is only partially shown: it indicates that this structure is appropriate when all the projects the user is interested in belong to the same domain of application (e.g., health, mining, etc.). (In the real system, a description of the domain would also be included as background information, and, if projects belong to different domains, they would be grouped and domains introduced appropriately. This is not done in our simplified example here.)

**Producing Discourse operators.** Once the content structure is defined, discourse operators can be automatically produced. During that process, scopes defined for content nodes will be encoded as constraints for discourse operators while retrieval service specifications will be embedded.

**Generating Brochures.** Now we call our Myriad platform with the generated discourse operators. These retrieval services are executed at runtime with the appropriate project ID from the current user model. As a result, the right content for the right project is retrieved. Figure 3 shows 2 examples of generated (simplified) brochures. We cannot describe here a detailed description of the adaptation mechanisms, the presentation strategies nor the space constraints. Nevertheless, for

the sake of completeness, a brief account is given below. Adaptation in the Myriad platform is achieved mainly through the scoping (constraint) mechanism available for both discourse and presentation planning. This flexible mechanism enables the specification of constraints for all the operators (discourse and presentation). Constraints are typically defined on both the data itself (e.g., availability, content and structure) and the available contextual models (including the domain, the user, the task, the environment and the discourse history). This means that, at run time, depending on the exact context, different operators will become applicable, and the system will adapt the generated document to the context (see [4] [14]). Other mechanisms also contribute to the adaptation of content: the variable binding and retrieval services, which ensure that the content of the generated document always reflect the current underlying data, and the space constraint algorithm, which reasons about the generated discourse and the device or environment model to ensure that the realisation of the discourse is appropriate for the device/environment under consideration.

## 4 Conclusions

Adapting information to users' special circumstance is desirable. However, developing adaptive systems is not an easy task. One of the challenges in building natural language generation based adaptive systems is to author the required discourse operators. One difficulty is that the expertise required to write them is complex. To address this issue, we have introduced a new approach to specify the discourse operators through the use of content structure, decoupling the specification of the structure of text from how to retrieve the data, and shielding authors from technical details of discourse operator encoding. We also presented a graphical tool, Constructor, to support authors in defining content structures. Content structures defined in Constructor are then transformed into discourse operators to be used by Myriad, our platform for the development of adaptive systems. There are two notable advantages over conventional discourse operator authoring: (1) In effect, people knowledgeable about the application domain are able to author discourse operators; and (2) the laborious encoding of discourse operators is replaced by a more intuitive visual composition of content structures. As a result, we expect that modifying discourse operators is an easier task when using Constructor.

Of course, there is still work to be done on Constructor, both in terms of its usability and its evaluation. We respect to the latter issue, our current plan is to carry out experiments where test subjects will use Constructor to design content structures for Scifly or a new application. Our aim will be to find out how usable and effective Constructor is in supporting non-technical experts to design new documents, identifying problems in the current implementation and in the proposed approach. Results will be used to improve the system. Finally, we would like to (1) examine how authors can also be supported in writing the required presentation operators and (2) look into an appropriate workflow model, involving domain experts, end users, and software engineers in the development of natural language generation based adaptive systems.

## Acknowledgments

We would like to thank the anonymous reviewers and Nathalie Colineau for their comments on the original version of this paper, and all the members of our team for their comments, suggestions, feedback and help on the tool itself.

## References

- [1] Androutsopoulos, I, Oberlander, J., and Karkaletsis, V.: Source Authoring for Multilingual Generation of Personalised Object Descriptions. *Natural Language Engineering*, Cambridge University Press (accepted for publication).
- [2] Brusilovsky, P.: Developing adaptive educational hypermedia systems: From design models to authoring tools. In: T. Murray, S. Blessing and S. Ainsworth (eds.): *Authoring Tools for Advanced Technology Learning Environment*. Dordrecht: Kluwer Academic Publishers, 377-409 (2003).
- [3] Campbell, M.K., DeVellis, B.M., Strecher, V.J. Ammerman, A.S., DeVellis, R.F. and Sandler, R.S.: Improving dietary behavior: The effectiveness of tailored messages in primary care settings. *American Journal of Public Health*, 84:783–787 (1994).
- [4] Colineau, N., Paris, C. and Wu, M. Actionable Information Delivery. *Revue d'Intelligence Artificielle (RSTI – RIA)*, Special Issue on Tailored Information Delivery, 18(4): 549-576. (2004).
- [5] Cristea A: “Authoring of Adaptive Hypermedia; Adaptive Hypermedia and Learning Environments”, book chapter to appear in "*Advances in Web-based Education: Personalized Learning Environments*", Sherry Y. Chen and Dr. George D. Magoulas. IDEA publishing group.
- [6] De Bra, P. and Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. In P. Brusilovsky and M. Milosavljevic (eds.), *The New Review of Hypermedia and Multimedia 4*, Special Issue on Adaptivity and user modeling in hypermedia systems, 115-139 (1998).
- [7] De Bra, P., Houben, G. J., and Wu, H. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *the Proceedings of ACM Hypertext*, 115-139 (1999).
- [8] Edmunds, A. and Morris, A.: The problem of information overload in business organizations: A review of the literature. *International Journal of Information Management*, 20(1):17–28, (2000).
- [9] <http://www.cebit.com.au/>
- [10] <http://www.csiro.au/scifly>
- [11] Lu, S., Paris, C. and Wu, M.: Document modelling for customised information delivery. In the *Proceeding of The Tenth Australasian Document Computing Symposium (ADCS 2005)*, Sydney, (2005).
- [12] Mann, W.C. and Thompson, S.A.: “Rhetorical Structure Theory: Toward a functional theory of text organisation”, In *Text 8* (3), pp. 243-281 (1988).
- [13] Moore, J. and Paris, C. Planning Text for Advisory Dialogues: Capturing Intentional and Rhetorical Information. In *Journal of Computational Linguistics*; 19 (4), December 1993. pp 651 – 694 (1993).
- [14] Paris, C., Wu, M., Vander Linden, K., Post M. and Lu, S.: Myriad: An Architecture for Contextualized Information Retrieval and Delivery (2004). In *the Proceedings of AH2004: International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*. August 23-26 2004, The Netherlands. Pp 205-214 (2004).