A Language for High-Level Description of Adaptive Web Systems

S. Hossein Sadat K. Mohtasham and Ali A. Ghorbani

Intelligent and Adaptive Systems Group
Faculty of Computer Science, University of New Brunswick
Fredericton, NB, E3B 5A3, Canada

Abstract

This paper focuses on the proposal, design, and implementation of AWL, the Adaptive Web Language. Also, an example application named PENS is explained and implemented in AWL. AWL development was inspired by several issues and shortcomings in the development of adaptive Web systems using the framework for adaptive Web systems, developed in the Intelligent and Adaptive Systems Research Group, at the University of New Brunswick, Fredericton, Canada. Lack of verification mechanisms and difficulty in development are two of the existing issues in the framework. Not only does AWL address those issues in the framework, but it also offers mechanisms to increase software quality attributes, especially, reusability. AWL has been designed based on the analysis of adaptive Web systems, having taken into account the principles of reuse-based software engineering (product-lines), domain-specific languages, and aspect-oriented programming, all of which are ongoing research fields in software engineering research. A compiler, named *AWL Compiler* has been developed, as the implementation of AWL for our adaptive Web framework. AWL Compiler automates the development process through hiding the framework internals from the application designer, and provides verification services so that applications can be verified to be consistent and meaningful. PENS, a personalized e-News system, is explained and its various aspects are developed using AWL.

Key words: Adaptive Web Systems, Domain-Specific Languages, Reuse-based Design

1 Introduction

As the World Wide Web is getting larger in size and more complex in structure, it has become more crucial for websites to guide the users through to relevant information, and present the information in an appropriate format. The users, in current Web, are usually overwhelmed with the huge amount of irrelevant information (and

links to information), that virtually makes them unable to navigate to find what they were looking for in the first place. In adaptive Web literature this is referred to as "lost-in-hyperspace" problem [1]. Furthermore, conventional websites provide the same information for any type of user at any situation (context), regardless of their background, knowledge, goals, and other context information. This "one-size-fits-all" problem [1] leaves some users with less knowledge or interest in a subject stranded and lost, while the other users become frustrated from seeing the same things they already know or are not interested in.

Relevant information can be fetched, filtered, and presented taking into account various *context* information. The user's preferences (interest, background, knowledge, etc.) and browsing behavior are part of the context information, which are referred to as *user model*. The environment-related parameters such as the location of the user, or the time, are considered as additional context information. Finally, technology-related parameters, such as client device characteristics and network bandwidth, are the other important elements of the context in an adaptive Web system. Adaptive Web systems use the context information to tailor the response of the system to the user's request through three different types of adaptation: content adaptation, navigation adaptation, and presentation adaptation. While there is some overlap between these types of adaptation, they are different in what they target. Content adaptation adds and/or removes information fragments to/from the page based on the current context. Navigation (structure) adaptation adds, removes, hides, sorts, and changes the color of the links in a page, in order to provide the best navigation structure in the current context for the user. Adding a recommended item at the end of a page, falls in this class of adaptation. Finally, presentation adaptation reformats the information fragments to achieve the appropriate final presentation for the current context. Resizing the images for a mobile device falls into this kind of adaptation.

The *adaptive Web systems framework* has been developed, in Intelligent and Adaptive Systems Group (IAS) at the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada, to provide the required platform for building adaptive Web systems. The framework provides components and protocols that allow a programmer to create adaptive Web systems in different domains such as educational, e-News, etc.

AWL, a language for adaptive Web systems, is proposed, designed, and developed to address the above issues. AWL is a domain-specific programming language, focused on the adaptive Web systems domain. AWL is the result of a reuse-based software engineering perspective that views the adaptive Web systems as a family of related products. Hence, the commonalities are considered to be realized by reusable components and the variabilities are supposed to be specified for each product. This is the software product-line approach to development, which increases productivity to a great extent.

The adaptive Web systems framework is supported through AWL Compiler, which has been developed to translate AWL programs into the framework's components and descriptions. Currently, AWL Compiler is functional and can generate adaptive Web applications for the framework in an easy, productive, and reusable form, as exemplified in Section 6.

One valuable facility that is provided by AWL is the provisions for *domain library* development. A domain library is a set of reusable artifacts and models, developed in AWL and targeting a specific domain within the adaptive Web domain such as e-News, e-Tailer, or e-Learning domains. Domain libraries maximize reuse: libraries can be developed for each domain, so that when new applications are to be built within that domain, many artifacts can be reused and extended. AWL has been designed to provide facilities for both library development and application development within the adaptive Web systems domain.

This paper is organized as follows. Section 2 is a summary of the concepts of adaptive Web systems, as well as a review of the past research in the area. Section 3 analyzes adaptive Web systems, as a family of related systems, to extract their common features as well as their variabilities. The details of AWL language are given in Section 4. Section 5 explains the AWL Compiler. First, the framework is introduced. Then, the various compiler parts are discussed through the use of UML diagrams. Section 6 presents PENS (Personalized Electronic News System) and explains its implementation in AWL.

Section 7 concludes the paper and suggests some future work directions.

2 Background Review

Adaptive Web systems (AWS) are systems that can adapt their features such as, presentation, content, and structure, based on the user's behavior and preferences, device capabilities, and environment attributes. Various models are used to describe an AWS. Three most common of these models are domain model, user model, and adaptation model [2]. The domain model describes the concepts, their relationships, and the way information is connected to the concepts. One may define domain model as consisting of content model (information model), concept model, and their relationships. The user model abstracts users' behavior and preferences to be used by adaptation model. The adaptation model describes *when* and *how* the system adapts.

2.1 Domain Model

The domain model describes the concepts and their relationships in the application domain as well as the way information is represented by the concepts. In fact, any application domain has a set of concepts specific to that domain, and well known to the experts in the domain. For instance, an e-News system domain has news item, report, and news category as its concepts.

Each domain has its own specific concepts and relationships. Therefore, creating new domain models has been an important issue in most of the works. For instance, AHA 2.0 introduces new visual tool to simplify the process of creating new concepts and relationships [3]. The visual specification then translates the drawing to XML specifications and adaptation rules. XML-based representations have been widely used to express concepts and relationships (ontology) in various works [4].

2.2 Presentation Model

The presentation model in an AWS describes the layout of the generated Web pages as well as the graphical style and attributes of the elements in them. When designed statically, the author of a Web page considers many visual factors and uses proper styles; whereas, when the pages are generated dynamically, the designer's artistic influence is absent. In order to minimize the inconsistencies in the dynamically generated pages, templates are used to enforce the layout and style of a group of pages. Although templates are very effective to enforce most of the page presentation attributes, they do not offer enough flexibility for page synthesis in AWS. It is desirable to go beyond templates and have a presentation description which not only allows the author to enforce the desired presentation through a top-down (template) description, but also provides the flexibility to support bottom-up (constraint-based) presentation description for highly dynamic pages. By bottom-up description we mean an approach in which a layout is defined hierarchically by other elements (which may have layout themselves) and the constraints and relationships between the elements of the layout are defined in all levels of hierarchy, and at runtime, from the lowest level to the top, the relationships are resolved and the final layout is generated. Using this approach, the author can both enforce a presentation template and define context-dependent relationships and presentation constraints. Nora [5] defines presentation model as the description of *where* and *how* navigation objects and access primitives will be presented to the user, and proposes a process for modeling the presentation. Although there are quite a few presentation languages that could be used to express a presentation, there are still some works proposing new languages for adaptable presentation modeling [6].

2.3 User Model

The user model abstracts users' behavior and preferences to be used by AWS. User model generally consists of two parts: demographic information (such as name, age, address) and application-specific part (such as knowledge, interest). In some systems, a table of attribute-values represents the user model [7]. Each attribute-value belongs to a page or concept. For example, the user's knowledge about the domain concepts is represented by these attributes. In this case, the user model is called an *overlay* user model. Many systems use overlay user models for their simplicity and effectiveness [7–10].

2.4 Adaptation Model

Adaptation can be based on different information such as user model, environment, and technology [11]. Adaptation can affect different aspects of the application: content, navigation, and presentation [12].

- *adaptive content* consists of selecting different information fragments (eg. text, images) based on the adaptation dimensions (user model, environment).
- *adaptive navigation* manipulates the possible navigation paths a user can take as well as the number, order, and target of the links.
- *adaptive presentation* deals with different layouts, and graphics features that may be suitable for current context.

Different users in different contexts need to see different content. The goal of content adaptation is to tailor information and services for users with respect to the context of the application. The content may be changed in different ways among which are: adding explanations, hiding/showing parts of information and natural language generation.

Different techniques have been used to do content adaptation: conditional fragments [7] and page variants, which are based on static content.

The goal of adaptive navigation is to prevent the user from getting lost in the hyperspace of the application. The methods used are

- Global guidance and orientation: assist the user to find the shortest path to the information she is looking for, and also to have a better view of the hyperspace and her position in it.
- Local guidance and orientation: assist the user in just one navigation step, that is, the best link to follow from current page.
- Personalized views: provide the user with a customized presentation of the application based on her preferences.

The different techniques that are used to accomplish the above methods are direct guidance, link annotation, link removing, link sorting, passive navigation, and map-adaptation. The navigation structure is usually determined by the domain concepts and the relations between them. The link relations are concept-based. That is, there is a link between concepts not fragments related to those concepts.

There is a concept hierarchy which is static and will not change during application life cycle. On the other hand, there might be changes in the relations and attributes of the concepts, per user model. So, each user may have her own view (instantiations) of the domain structure. The objective of adaptive presentation is to provide different layouts and graphical attributes to different users in different contexts. Different adaptation methods include the ordering of information fragments, colors, font attributes and fragment size. The techniques used for content adaptation can also be used here. Presentation adaptation comes into play when the contents of a page and their link relationships are resolved. Then, the presentation of these contents to the user is adapted based on some criteria. For instance, one common scenario is when the user is using a handheld device. Because of the limited capabilities of the handheld devices, the presentation of the page should be changed, in several ways, such as image size and color depth reductions and page layout adjustments.

2.5 Dynamic Page Generation

A Web server produces a Web page in response to the user's request. This Web page is created either dynamically or already exists as a static file. Adaptive Web systems, like any other Web system, produce a page in response to the user's request. However, adaptive Web systems should change the response based on some adaptation dimensions, such as *user model*, *technology* and *environment* [11]. Therefore, the nature of these systems demands the dynamic creation of pages.

On the other hand, the dynamic generation of Web pages constitutes a continuum that defines dynamism with different degrees. At one end we have static page delivery, and at the other end totally dynamic page content, template, and appearance generation. There is no systematic procedure for this dynamism. Each application may end up with some degree of dynamism.

With all this in mind, and also considering the fact that adaptive hypermedia systems can be defined by three models: *User Model, Domain Model*, and *Adaptation Model* [13] , we are interested in formalizing and systematizing dynamic page generation based on the adaptive hypermedia system models. In other words, we want to develop an engine that produces a page based on the current models of the system. We refer to this engine as *Synthesizer* and the corresponding process is called *Synthesis*.

According to the American Heritage Dictionary, synthesis is "the combining of separate elements or substances to form a coherent whole." In the adaptive Web systems context, the "elements" and "substances" are *information fragments* (we refer to them as *components* too) and the "whole" is the final Web page.

There are not many references to page synthesis in the adaptive hypermedia literature (see [1] for a review of adaptive hypermedia systems). Most of the systems use available pages and remove or add some elements to adapt to the user (*e.g.* AHA! [7,14–16]). However, some systems generate the pages dynamically based on the user model (*e.g.* SETA [17]). In this section, some related projects that apply some degree of synthesis are reviewed.

Index page synthesis [18], is one of the works that directly deals with the synthesis problem. In this work, page synthesis is defined as the automatic creation of Web pages. An index page is a page consisting of links to a set of pages that cover a particular topic. The synthesis of a new index page is divided into the following sub-problems:

1) What are the contents of the index page?
2) How are the contents ordered?
3) What is the title of the page?
4) How are the hyperlinks on the page labeled?
5) Is the page consistent with the site's overall graphical style?
6) Is it appropriate to add the page to the site? If so, where?

In [18], only the first sub-problem is investigated. The rest of the problem remains to be investigated. Besides, the page to be synthesized is an index page, not a general page with variant layout and style. SETA [17] is a prototype toolkit for building adaptive Web stores. It dynamically generates the pages of a Web store catalog and selects the content of the pages based on the user's interests and familiarity with the products. Also, the system sorts the available items for a product class based on the user's preferences (See [19,20] for more details). SeAN [21] is an adaptive system for personalized access to news. It uses a structured hierarchy to represent news (domain model). SeAN has three goals: first, to select news topics relevant to the user. Second, to present an appropriate level of details of the news based on the user model and third, to provide advertisement most relevant to the current page and the user.

Peba-II [22] is an on-line animal encyclopedia that produces descriptions and comparisons of animals as WWW pages. The Peba system contains information about hundreds of animals. The system produces descriptions and comparisons of animals by using an underlying representation of information, which is similar to a database. The Web pages presented to the user do not exist on the server, but are dynamically created when you ask for information.

2.6 Domain-Specific Languages (DSL)

A domain-specific language is considered as the final phase of the evolution of an application framework [23]. An application framework realizes the necessary domain architecture and components, and a DSL, using the mature domain knowledge, provides a systematic, reusable, and verifiable way of instantiating new applications within the domain. There has always been a need for more specialized language support to solve problems in well-defined application domains, to enable reuse and facilitate development. Several solutions have been practiced by developers to meet this need:

- Subroutine libraries
- Object-oriented frameworks and component frameworks
- A domain-specific language (DSL)

The first two approaches are quite common and we don't address them here. The third approach is the one we would like to use. "A domain-specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain." [23] According to this definition, the key characteristic of a DSL is its focused expressive power. VHDL hardware description language and SQL database query language are two examples of domain-specific languages, specific to the hardware design domain and database domain, respectively.

There are many issues related to DSL development. Here we bring some advantages and disadvantages of DSLs. There are risks and opportunities in developing and using a DSL. A well-designed DSL manages to find the proper balance between advantages and disadvantages. The advantages of a DSL are:

- DSLs allow the solution to be specified at the level of abstraction of the problem domain, hence, domain experts themselves can validate, modify, or develop DSL programs
- DSL programs are concise, self-documenting, and reusable
- DSLs enhance productivity, reliability, maintainability, and portability
- DSLs embody domain knowledge, hence, enable the conservation and reuse of knowledge
- DSLs allow validation and optimization at the domain level
- DSLs improve testability

Although these benefits justify developing a DSL, there are some disadvantages that should be taken into account the most important of which are: the high cost of designing, implementing, and maintaining a DSL, and the difficulty of balancing between domain-specificity and general-purpose programming language constructs.

3 Reuse-Based Design of Adaptive Web Systems

In this section, we analyze adaptive Web systems with a product-line-engineering perspective to develop and justify the required basis for moving toward an adaptive Web system specification language.

3.1 Adaptive Web Domain Engineering

A family of applications (products) is referred to as a *domain*, and domain engineering is the set of activities aiming at generating reusable assets across a domain. In domain engineering, the domain expertise and knowledge are organized to develop a framework of reusable components and artifacts, which can be instantiated to generate products within the domain. Domain engineering includes two major processes: domain analysis and domain implementation. Domain analysis is the process of gathering, organizing, and modeling information about the products within the domain. In domain implementation, an architecture is designed to support the domain, based on the results of domain analysis phase.

3.1.1 Domain Analysis

In domain analysis, the scope of domain is determined and the terms of the domain are defined. Domain analysis includes several activities (see [24]), however, its main activity is *commonality analysis* whose outcome is the commonalities and variabilities among the products of the domain.

We define adaptive Web systems family (domain) as all Web-based software systems that adapt their response to the users, based on the current context information, such as user browsing behaviour, environment, and device capabilities. Although in this definition, all Web-based systems are considered as potential systems to be adaptive, adaptive Web systems techniques are most effective for information providing systems, or *online information systems*, where the main purpose of these systems is to present useful and relevant information to the user.

Since the notions and concepts of adaptive Web systems have been well defined and explained in [5], they are not elaborated here; instead, the new features and the family-based approach are emphasized.

¿From the usage perspective, adaptive Web systems share a common general use-case.They provide relevant information, from different sources, to users. That is, when the user asks for a specific *concept*, the system provides the information on that concept from information sources, and tailors the information based on the context of the request, which includes the user's preferences, browsing history, and

so on. From this common ground, the basic commonality of AWSs can be extracted: all AWSs constantly run the following high-level algorithm.

Algorithm 1 High-Level LOOP in AWS

```
WHILE (there is a request) {
   Receive the request
   Extract the relevant context information
   Update context information
   Compose and send the response
}
```

In fact, most Web-based systems share part of this commonality; they provide information as the users' requests arrive. However, what is specific to AWSs is a systematic way of using and updating context information to keep up with the users' changing interests, needs, and other conditions. Once a request is received from the user, the requested concept, which abstracts and represents a piece of structured information, is extracted from the request. Then, the system queries the information sources (that has been designated as information providers for to the requested concept) to gather and structure the information representing the concept. These queries are formed taking into account various context information. For instance, if a user connecting from Canada is requesting for national news headlines, an adaptive e-News systems would provide all headlines that are related to the location of the user (which is a piece of context information); so, the user would see Canada's national news headlines.

After the information is fetched, there is still additional context information that might be used to tailor the information presentation. If the user is connecting using a hand-held device, for instance, the format and style of the information should be adapted to be presentable by the device, considering the limitations, capabilities, and preferences of the device, all of which form a part of context information. Finally, the presentation is composed and sent back to the user.

Commonalities are abstracted, designed, and implemented as reusable components to be reused within adaptive Web systems domain. Hence, when designing new applications, commonalities are not implemented again. They are automatically integrated into the applications. This integration mechanism is supposed to be simple and transparent, which is a major goal of the family-based design of the AWSs. Domain implementation realizes these common assets and components, and its purpose is to design, implement, and test the components (within a framework) to be reused later on through instantiation, parameterizations, or specialization.

3.1.2 Domain Implementation

Figure 1 shows the architecture of the framework that implements the domain of adaptive Web systems. This architecture has been designed considering the impor-

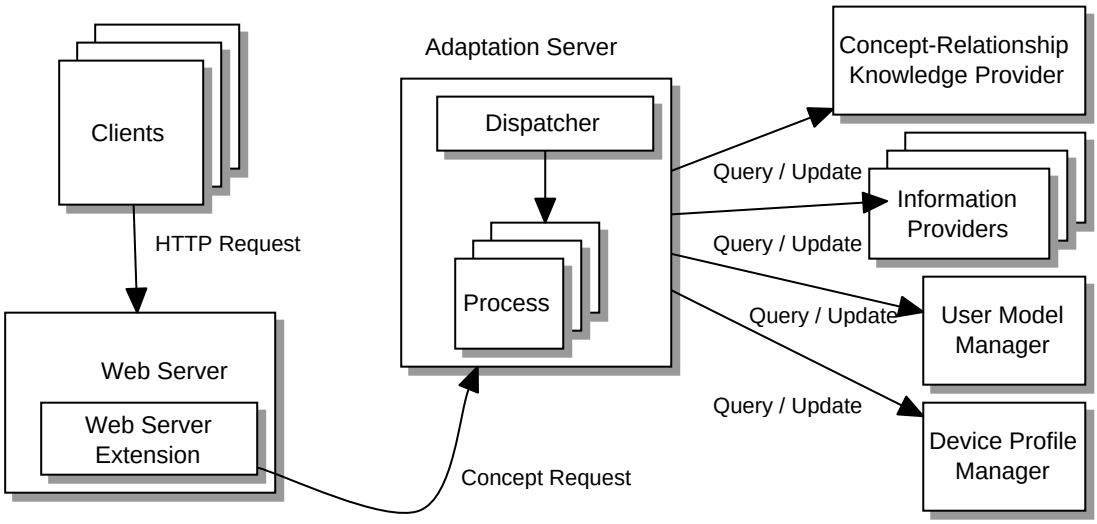


Fig. 1. The Architecture of Adaptive Web Systems Framework

tant observations from the domain analysis process. These observations as well as the architectural component that realizes the observation are as follows:

- Clients communicate with the AWSs through the HTTP protocol. Hence, a centralized HTTP request processing component (Front Controller [25]) receives all clients' requests and creates a more informative request for the system.
- All requests go through the same process; therefore, an adaptation server is designed to be responsible for processing individual requests, querying different information sources, and forming the final response.
- The framework needs a way of keeping users' personal information, preferences, and past (browsing) behaviour. This is realized by providing a dedicated component for user model management.
- Users might be using various devices to connect to the AWS. The framework needs facilities to recognize different devices and extract their characteristics, limitations, and capabilities, in order to provide the best presentation adaptation. Device Profile Manager provides these services.
- Adaptive Web systems might use existing domain knowledge and information structure in terms of concepts and the relationships between them, to better provide related information. There should be a component that can store, manage, and provide this information efficiently and easily. Concept-Relationship Knowledge Provider is responsible for this important functionality.
- AWSs provide information to their clients. The information might be scattered across various sources. There need to be components providing services for querying and retrieving information from these sources; Information Providers are such components.

3.2 Adaptive Web Application Engineering

Using a framework for adaptive Web systems, through which commonalities have been realized, developed, and integrated in the form of reusable components, developing new applications should ideally be the process of specifying the variations; that is, the aspects which make an application different from others within the domain. In domain engineering, the commonalities were implemented to produce reusable components. In application engineering, on the other hand, the variations are expressed using specification tools and models. Variations are the features, functionalities, or qualities that vary from application to application in a family. In AWS family, variations are specified by answering to these questions: how presentations are organized and styled? What are the information structure and sources? How is the user modeled and kept track of? When and how adaptation takes place?

In order to avoid confusion, whenever the term "domain" is used in the rest of this paper, it refers to one of the application domains within adaptive Web systems domain, unless specified otherwise (the "domain analysis" and "domain engineering" titles refer to the adaptive Web systems software domain, not to be confused with the application domains, such as e-News and e-Tailer, just explained above).

Within the adaptive Web systems domain, there are application domains with a common set of concepts, information structures, abstractions, and functionalities. For example, adaptive Web-based course tutoring (educational) domain features online course materials and facilities that, through using user's knowledge and background, make learning easier for the users. E-Tailer domain includes systems that provide personalized recommendation of products and services, based on the user's interest, history of browsing and shopping, and other criteria. E-News domain includes news presentation systems that, based on various context information, provide users with the most related and interesting news items. There are other domains such as online books, libraries, online museums, encyclopedia that can generally be referred to as online information systems (See [26] for a list of adaptive Web systems and their domains).

Our approach in the design and implementation of adaptive Web systems is a reuse-based approach. Hence, whenever there is the opportunity to apply reuse principles, they are applied. Hence, not only was the domain engineering a reuse-centric process, but also the application engineering applies reuse extensively. That is, for all the variations that should be specified to develop new applications, it is desirable to abstract and specify them in a reusable fashion. For instance, in e-News domain, the domain library developer (e-News domain developer or designer) may assume that all e-News applications, no matter what functionality they provide, use some common terminology, concepts, presentation format, and even adaptations. Therefore, these common feature might be abstracted and implemented in a reusable way. In order to develop an e-News application, then, these reusable elements are cus-

tomized, instantiated, or just used without any change. It is not a crisp decision to extract identify some elements as reusable ones; it totally depends on the designer of the system.

Here is application engineering process proposed based on the approach explained in this section:

1) Application requirements elicitation and analysis (as traditional software engineering processes).
2) Extracting and abstracting the information structure in terms of concepts and their relationships.
3) Modeling users of the system through the design of a user model; this includes selecting those attributes of the user that may be used to provide more relevant information, such as demographic as well as dynamic behavioural attributes.
4) Developing the necessary information provider components to support information retrieval for the application. If a domain library has already been developed that contains such components, then the components are just reused.
5) Specification of presentation templates, including the hierarchies of fragments, their styles, and the information they present.
6) Adaptation specification, including content, navigation, and presentation adaptations (as defined before).

This process can be applied to a domain such as e-News, e-Tailer, etc., or it might be applied to an application within a domain. In either case, the same process is followed with one difference. When applying the process to design a domain library, reusability is the main concern (designing reusable assets); however, when applying it to an application, reusing the existing domain library (reusable elements) is to be practised in all phases of the process.

3.3 Towards a Domain-Specific Language for Adaptive Web Systems

In previous sections, a family-based approach to adaptive Web systems development was presented that applies *reuse*, as its main principle, to all phases of development. The results of such reuse-based engineering is a framework with reusable components; however, there is no mechanism that enforces the reuse of these components or the way they are supposed to be reused. No readily-available process exists to verify the way these component are used within an applications. In other words, a framework is a step forward toward reuse, but, how reuse is supposed to be applied by application engineer, is not enforced or provisioned within the framework. Therefore, in order to meet the goals of reuse-based software engineering in the domain of adaptive Web systems, we need a mechanism that, based on the underlying (existing) framework(s), provides reuse mechanism in a systematic, hence, verifiable process.

A domain specific language (DSL), for adaptive Web systems domain, is the means to enforce reuse within the domain. In fact, "a DSL is viewed as the final and most mature phase of the evolution of an object-oriented application framework." [23]

A DSL brings many other important advantages to our AWS development framework. A DSL, designed for AWSs, has expressive power focused on AWSs. Therefore, it allows the solution to be specified at the level of abstraction of the problem domain; that is, the user of the DSL only deals with the concepts within the AWSs domain. Hence, domain experts themselves can develop. validate, and modify DSL programs. Such a DSL enhances productivity, reliability, maintainability, and portability, all because of its simple, reusable, and domain-specific structure that automatically reuses the framework assets. Furthermore, a DSL allows validation and optimization at the domain level.

Although the advantages are tempting enough to inspire us to develop such a language, it should be noted that these advantages do not come for free. The cost of designing, implementing, and maintaining a DSL is high; especially, the designer has to have both domain knowledge and programming language and compiler development expertise.

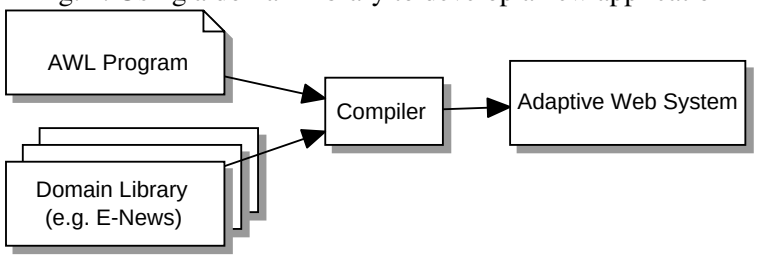
4 AWL: Adaptive Web Language

An adaptive Web system framework can provide reusable components to be used for creating adaptive Web systems. In fact, creating a new application using the framework would be much simpler than creating one from scratch, since many components and protocols (which are supported through the framework) do not need to be implemented. Nevertheless, although a framework supports reuse-based software development by providing reusable components, the framework user, or rather, the application programmer remains responsible for actually reusing the components, which makes application development not only hard but also error prone. The application programmer has to be familiar with the general purpose programming language that the framework uses; (s)he has to know the framework and how it works in detail; and (s)he has to decide what components to use at what circumstances.

The best tool that can support a framework in a systematic fashion is a domain-specific language (DSL) designed specifically for the framework. Such a DSL can hide the framework's internals from the application programmer. The programmer can use the DSL to create new applications, knowing only the DSL's vocabulary and constructs, which are usually the same as the framework's domain vocabulary.

In the following sections, a language for the adaptive Web systems framework (AWL), which is a DSL focused on adaptive Web systems, is proposed. In Sec-

Fig. 2. Using a domain library to develop a new application



tion 4.1 the language requirements are elaborated. Then, in Section 4.2, the syntax of AWL is presented and Section 4.3 explains the semantics of AWL. Section 4.4 proposes a development process when using AWL. Two AWL design perspectives are discussed in Section 4.5.

4.1 Language Requirements

AWL should provide abstractions, facilities, and constructs that allow the designer and/or the author of an AWS to specify various aspects of the AWS that are variations not commonalities, according to the adaptive Web systems family architecture. In addition to the variations, the language should provide necessary means for reusable modeling. That is, even at the time of reusing framework's components (through the language), reusability still remains an important concern. Reusability at the language level can be best achieved through a library-of-reusable-code strategy: adaptive Web systems fall into different categories; for instance, e-News systems, e-Tailers, and etc., each of which is called an application domain (as pointed out in the previous section, this shouldn't be confused with "domain" when used for AWS family-based development, since it refers to the adaptive Web domain.)

Different applications within these domains share a lot in common. Hence, based on reuse principles, it is desirable to model the commonalities as reusable assets in a *library*, to be used for the applications within the domain (Figure 2). A DSL for AWS should provide facilities for both library (domain) development and application development. That is, if one wants to design an e-Tailer, if there already exists an e-Tailer library implemented in the language, then the library is used along with other extensions, instantiations, and customizations to create the application. Otherwise, everything should be implemented from scratch.

Based on the architecture presented for AWS and the above discussions, the following requirements are extracted to be met by the language:

- The language should provide some mechanisms for abstracting and modeling information structure, in terms of concepts and relationships. These mechanisms should include both information modeling and querying.
- The language must provide necessary means for hierarchical presentation speci-

fication, which includes both layout and formatting definitions.

- The language should be independent from the final (generated) presentation language; that is, the programs written in the language might be translated to various concrete presentation outputs (*e.g.* HTML), hence, the language structure should not be affected by any target presentation output.
- The language must provide user model specification constructs, so that authors will be able to design custom user models that suit a specific target domain.
- The language must provide the necessary constructs, operations, and abstraction facilities to to express adaptation. Adaptation must be expressible in a separate model. This requirement implies that if the adaptation model is removed from an application, then the result would be a non-adaptive version of the application.
- The language should allow reuse of existing specifications, such as presentations, user model, and adaptation models, through inheritance and parameterizations.
- The language should provide other handful modularization facilities (functions, for instance) to allow authors to produce a more reusable code.
- The language should be usable at two levels: library level, and application level. This requirement comes from the fact that there might be two users with different programming capabilities and domain expertise. For example, for designing an e-News Web application, a designer/programmer who has a quite good knowledge of both programming and the domain of e-News, analyzes, abstracts, and codes a set of reusable components, including concept and relationships (information structure), user model, presentations, and even adaptation specifications, and produces an e-News domain library. Afterwards, an author, who is expected to have less expertise in programming, uses the domain library to create an application (a specific e-News Web system). In this way, the author just uses the already designed elements of the domain, although (s)he can extend or overwrite them.
- The language should be simple enough to be learned, in a fairly short time, by ordinary Web developers.

4.2 Language Syntax and Program Structure

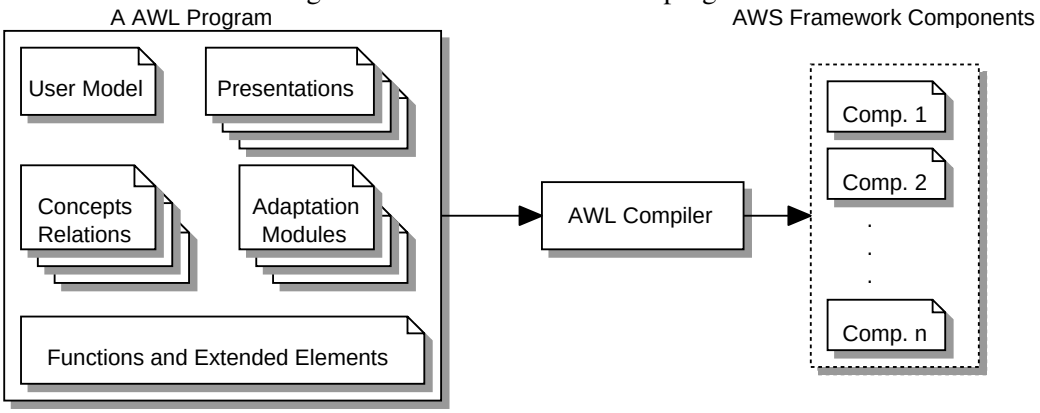
The syntax of a language determines the structure of the acceptable programs in the language. When designing the AWL syntax, usability, simplicity (writability, and readability of programs [27]), and separation of concerns (as the main principle of aspect-oriented programming [28]) were the main quality attributes taken into account. The EBNF [29] syntax of the language is as follows:

(1) <AWS> ⇒ {<CONCEPT> | <RELATIONSHIP> | <PRESENTATION> | <USERMODEL> |
<ADAPTATION> | <EXTENDED-ELEMENT> |
<FUNCTION>} [<INSTANCE-MODEL>]
(2) <PRESENTATION> ⇒ [<PRES-MODIFIER>] 'fragment' <ID> ['extends' <ID>] ['realizes' <ID>]
'{'{(<ITEM> | <ITEM-LIST> | <ATTRIBUTES> | <PROPERTY>)} [<INIT>] '}'
(3) <PRES-MODIFIER> ⇒ 'main' | 'page'
(4) <ITEM> ⇒ 'item' <ID> ':' <FRAGMENT-TYPE>
'{' {<ITEM-SPEC>} '}'
(5) <ITEM-LIST> ⇒ ('verticalBox' | 'horizontalBox') '('<EXPR-LIST>')' 'item' <ID> '['"$"']' ':'
<FRAGMENT-TYPE> '{' {<ITEM-SPEC>} '}'
(6) <FRAGMENT-TYPE> ⇒ ('Image' | 'Text' | 'XHTML' | <ID>) ['(' <EXPR-LIST> ')']

(7) <ITEM-SPEC> ⇒ (<ASSIGNMENT> | <ITEM-PROP>) ';'
(8) <ITEM-PROP> ⇒ ('linksTo' | 'top' | 'bottom' | 'rightmost' | 'leftmost' | 'left' | 'right' | 'above' | 'below' | 'bold' | 'underlined' | 'italic' | 'large-FontSize' |
'smallFontSize' | 'strongText' | 'paragraph' | 'emphasized') [<ENTITY-NAME> ['(' <EXPR-LIST> ')']]
(9) <PROPERTY> ⇒ (<ALIGNMENT> | <MARGIN> | <HEADER>) ';'
(10) <ALIGNMENT> ⇒ 'align' ('left' | 'right' | 'top' | 'bottom') [':' <ID> {',' <ID>}]
(11) <MARGIN> ⇒ ('marginX' | 'marginY') '=' <EXPR>
(12) <HEADER> ⇒ 'header' ':' { <ID> <EXPR> ';'} 'header'
(13) <USERMODEL> ⇒ [<USER-MODIFIER>] 'usermodel' <ID> ['extends' <ID>] '{'{<USER-ATTRIBS> | <STRUCT-DEF>} ['events' ':' {<EVENT-DEF>}] [<INIT>] '}'
(14) <USER-MODIFIER> ⇒ 'main'
(15) <USER-ATTRIBS> ⇒ ['overlay' '('<ID> ')'] <ATTRIBUTES>
(16) <STRUCT-DEF> ⇒ 'define' <ID> '{' { <ATTRIBUTES> }'}'
(17) <EVENT-DEF> ⇒ 'on' <ID> <PARAMETERIZED-BLOCK>
(18) <ADAPTATION> ⇒ [<ADP-MODIFIER>] 'adaptation' <ID> ['extends' <ID>]
'{' [{<ADP-PROP>}] {<ADP-TARGET-DEF> | <ADP-STATEMENT>} '}'
(19) <ADP-MODIFIER> ⇒ <ID>
(20) <ADP-PROP> ⇒ (<ASSIGNMENT> | <ID> [<ID>]) ';'
(21) <ADP-TARGET-DEF> ⇒ 'target' <ID> ['(' <ID>{',' <ID>} ')'] ':'
<ADP-TARGET-SET> ';'
(22) <ADP-TARGET-SET> ⇒ <PATTERN> { ('+' | '-') <ADP-TARGET-SET>} | '('<ADP-TARGET-SET> ')'
(23) <PATTERN> ⇒ <STRING>
(24) <ADP-ADAPT> ⇒ 'adapt' <ID> ['('<ID>{',' <ID>} ')']
'{' {<STATEMENT>} '}'
(25) <CONCEPT> ⇒ [<CONCEPT-MODIFIER>] 'concept' <ID> ['extends' <ID>] '{' { <ATTRIBUTES> | <RELATIONSHIP-DECLARATIONS> } [<INIT>] '}'
(26) <CONCEPT-MODIFIER> ⇒ <ID>
(27) <RELATIONSHIP-DECLARATIONS> ⇒ <MODIFIER> 'relationship' <ID> <ID> : <ID> {',' <ID> <ID> : <ID> } ';'
(28) <RELATIONSHIP> ⇒ <REL-MODIFIER> 'relationship' <ID>
'('<PARAMETER-LIST> ')' '{'<REL-BODY>'}'
(29) <REL-MODIFIER> ⇒ <ID>
(30) <REL-BODY> ⇒ <CLAUSE> {'or' <CLAUSE>}
(31) <CLAUSE> ⇒ '('<CLAUSE> ')' | <PREDICATE-CALL>
{'and' <PREDICATE-CALL>}
(32) <PREDICATE-CALL> ⇒ <ID> '(' <PARAMETER-LIST> ')'
(33) <EXTENDED-ELEMENT> ⇒ 'define' <ID> 'as' <TYPE>
'{'<EXTENDED-BODY> '}'
(34) <EXTENDED-BODY> ⇒ ('get' ':' | 'set' <PARAMETER-LIST> ':')
{<STATEMENT>}
(35) <FUNCTION> ⇒ 'define' <ID> <PARAMETERIZED-BLOCK>
(36) <INSTANCE-MODEL> ⇒ 'instantiation' '{'{<STATEMENT>}'}'
(37) <PARAMETERIZED-BLOCK> ⇒ '('<PARAMETER-LIST>')'
'{'{<STATEMENT>}'}'
(38) <STATEMENT> ⇒ (<ASSIGNMENT> | <EXPR> | <IF-STATEMENT> | <OPERATION> |
<INSTANTIATION> | <REL-QUERY> | <FACT>) ';'
(39) <FACT> ⇒ '.' <PREDICATE-CALL>
(40) <REL-QUERY> ⇒ '?' <PREDICATE-CALL>
(41) <INSTANTIATION> ⇒ 'instantiate' ['list' 'of'] <ID> [<LIST>]
(42) <OPERATION> ⇒ '!' <ID> ['('<EXPR-LIST>')'] ';'
(43) <IF-STATEMENT> ⇒ 'if' <EXPR> '{' {<STATEMENT>} '}' ['else' '{' {<STATEMENT>} '}']
(44) <ASSIGNMENT> ⇒ <ENTITY-NAME> '=' <EXPR>
(45) <ATTRIBUTES> ⇒ <MODIFIER> 'attribute' <ID> ':' <ATTRIB-TYPE> {',' <ID> ':' <ATTRIB-TYPE> } ';'
(46) <ATTRIB-TYPE> ⇒ 'realNumber' | 'string' | 'boolean' | 'date' |
'list' 'of' <ID> | <ID> | 'alternative' '{' <STRING> {',' <STRING> } '}'
(47) <INIT> ⇒ 'init' ':' <PARAMETER-LIST> '{' { <STATEMENT> } '}'
(48) <PARAMETER-LIST> ⇒ ['%'] <ID> ':' <ATTRIB-TYPE> { ',' ['%'] <ID> ':' <ATTRIB-TYPE> }
(49) <EXPR> ⇒ <CONDITIONAL-AND-EXPR>
{'or' <CONDITIONAL-AND-EXPR> }
(50) <CONDITIONAL-AND-EXPR> ⇒ <RELATIONAL-EXPR>
{ 'and' <RELATIONAL-EXPR> }
(51) <RELATIONAL-EXPR> ⇒
<ADDITIVE-EXPR> [<RELATIONAL-OP> <ADDITIVE-EXPR>]
(52) <ADDITIVE-EXPR> ⇒ <TERM> {('+' | '-') <TERM>}
(53) <RELATIONAL-OP> ⇒ '<' | '>' | '<=' | '>=' | '==' | '<>' | 'in'
(54) <TERM> ⇒ <FACTOR> {('*' | '/') <FACTOR>}
(55) <FACTOR> ⇒ '-' <FACTOR> | '('<EXPR>')' | <LITERAL> | <LIST> | <ENTITY-NAME> | <FUNCTION-CALL> | <PREDEFINED>
(56) <PREDEFINED> ⇒ 'userID' | 'sessionID' | 'clientIP' | 'URI' | 'browserType'
(57) <LITERAL> ⇒ <STRING> | <INTEGER-NUMBER> | <REAL-NUMBER> | 'false' | 'true' | '$'
(58) <ENTITY-NAME> ⇒ <BRACK-ENTITY> { '.' <BRACK-ENTITY> }
(59) <BRACK-ENTITY> ⇒ <ID> ['[' <SUBSCRIPT> ']']
(60) <SUBSCRIPT> ⇒ <EXPR> ['..' <EXPR>]
(61) <LIST> ⇒ '[' <EXPR-LIST> ']'
(62) <EXPR-LIST> ⇒ <EXPR> {',' <EXPR>}
(63) <FUNCTION-CALL> ⇒ 'call' [<ID> '.'] <ID> ['(' <EXPR-LIST> ')']
(64) <ID> ⇒ <LETTER> {<LETTER> | <DIGIT> }
(65) <INTEGER-NUMBER> ⇒ <DIGIT> {<DIGIT>}
(66) <REAL-NUMBER> ⇒ (<DIGIT>{<DIGIT>}'.' {<DIGIT>}
| '.' <DIGIT>{<DIGIT>})'r'
(67) <DIGIT> ⇒ ['0'-'9']
(68) <LETTER> ⇒ ['a'-'z'] | ['A'-'Z'] | '_'
(69) <STRING> ⇒ '/' { ~ ['/','\','\n','\r'] | ('\' ['n','t','b','r','f','\','/']) } '/'

Figure 3 shows the structure of an AWL program. All the modules inside a program are processed by AWL compiler and weaved to generate the necessary components and artifacts in the framework.

Fig. 3. The structure of an AWL program



4.3 Language Semantics

The semantics of a language is the meaning of the program elements and units, coded in the language. The meaning of language statements, expressions, and constructs can be used by compiler developers (to develop compilers for the language), or by programmers who want to program in the language. In this section, the concepts of adaptive Web systems domain are used to explain the AWL semantics. When AWL parser reads a part of a program that corresponds to a production rule in the syntax, it creates an object to represent the left-hand-side element in the production rule. This object keeps references to all the elements on the right hand-side of the production rule. Hence, at the end of parsing, the parser returns a tree whose root node represents the highest-level element in the syntax, *i.e.* <AWS>. This tree holds the structure of the parsed program in terms of language syntax elements. This tree is referred to as the *abstract syntax tree* of the program. However, the tree that is built in the above way is not only an abstract syntax tree, but also a semantic tree. In fact, the objects created for each intermediate node in the syntax tree (the intermediate nodes are those that are not leaf; in other words, they are left-hand-side elements of at least one production rule) have the meaning of the production rule embedded. In this section, the meaning of the production rules is informally explained.

Rule 1 defines <AWS>, which represents a program in AWL. When a program is parsed, the parser returns an object of the type <AWS>. <AWS> has methods and attributes for storing a program's meaning in the form of concepts (<CONCEPT>), relationships (<RELATIONSHIP>), presentations (<PRESENTATION>), adaptation modules (<ADAPTATION>), user model modules (<USERMODEL>), extended entities (<EXTENDED-ELEMENTS>), functions (<FUNCTIONS>, and instantiation model (<INSTANCE-MODEL>). This production rule defines an AWL program as containing a set of modules, each of which describes an aspect of the application. Concepts and relationships are used to model the domain, to which the application belongs (*e.g.* e-News, or e-Learning). Presentation specifi-

Table 1
Fragment styles that are set through assignment

Property	Example Values
color	‘blue’, ‘#E0EEEE’
bgcolor	‘blue, ‘#E0EEEE’
fontFace	‘helvetica’, ‘times’
textScale	‘100%’
scale	‘100%, ‘50%’
border	‘InFrame’, ‘Floor’
length (text)	‘300’
width	‘300’
height	‘200’

cations describe how information is presented to the user, using a hierarchical fragment structure. Adaptation modules abstract the adaptation strategies provisioned by the designer in the application. The user model module specifies the structure and dynamics of the user model. Functions and extended entities are not essential parts of the language, however, they provide the means for a designer to make domain libraries that are easy to use; hence, authors (that might not have a general-programming-language expertise) can easily express the application through custom domain vocabulary extended by the designer. The instantiation module can be used for configurations and one-time initializations. Currently, it is not being used in the language, but is reserved for later language extension.

The production rules 2 to 12 define a presentation module. A <PRESENTATION> may inherit from other presentations (through *extends* keyword), and it may also be the corresponding presentation description of a domain concept, in which case, the name of the concept would come after *realizes* keyword. A presentation is composed of a set of attributes (<ATTRIBUTES>), items (<ITEM>), item lists (<ITEM-LIST>), presentation properties (<PROPERTY>), and an optional initialization -parametrization section (<INIT>).

The presentation modifier (<PRES-MODIFIER>) can be used to mark a presentation as a *page* or as the *main* presentation module of the system, which makes the presentation the front page of the system (the first page the user sees when connecting to the system). The attributes (defined by rule 45) of a presentation define the information presented by the presentation module. They can also be used as variables that hold intermediate results during the information retrieval and processing. The items define the fragments inside a presentation, and are themselves defined through the production rule 4. An item has a name and a type. The fragment type can be *text*, *image*, *XHTML*, or a user-defined presentation. An item list defines an

array of fragments with the same type and properties (rule 5). The item lists can be vertical or horizontal in the presentation layout. The parameters that are passed to an item list may determine the list's size and ordering criteria (currently only size is supported). The item properties (rule 8) specify various formatting and style attributes of the fragment (Table 1 shows those style properties that are set by an assignment), as well as layout and hypermedia relations, such as *linksTo* relation that links a fragment to another presentation. The presentation can also have properties that specify margin, alignment, and header (if it is a page) information (rule 9).

The Production rule 13 defines a user model module. In a program, there can be as many user model modules defined as possible; however, only one can be the *main* user model, which can be specified through the user model modifier (<USER-MODIFIER>). The user model consists of a set of attributes; the attributes can have either primitive types (*string*, *realNumber*, etc.) or user-defined structures. An attribute in user model can be defined as an *overlay* attribute, meaning that there is not just one value associated with the attribute but a vector of values, each of which corresponds to a concept in the domain. The user-defined structures are defined inside the user model module. They are recursively defined as a set of attributes (rule 16). In addition to static structure of user model, the dynamic aspect of user model (how it is updated) needs to be specified too. *Events* are used to specify when and how the user model is updated. An *event* has a name, a parameter list, and a block of statements that can update attributes of user model (rule 17). For instance, the event corresponding to a page visit is named *visited* and it takes one parameter: the name of the visited page.

The production rule 18 defines an adaptation module (<ADAPTATION>). An Adaptation module is composed of a set of target definitions (<ADP-TARGET-DEF>) and adaptation statements (<ADP-ADAPT>). Each adaptation *target* specifies a set of presentation items, which will be adapted through adaptation statements inside the module. According to rules 21 to 22, a target has a name and a parameter list. The name is used in the adaptation statements to refer to the target set. The parameter list is not currently used. A target is recursively defined as the union or difference of other target sets (<ADP-TARGET-SET>). The recursion stops when the set is a pattern string. A pattern string is used to easily and effectively select presentation elements based on the item's name, type, or containing presentation. For instance, if one wanted to define target *allImages* to be the set of all images in all the presentations, the target definition would be written as follows.

```
target allImages: 'item *:image';
```

The wildcard character means any match. The type of the items is specified after ':' in the pattern string. If all the images inside *Banner* were to be selected, the following pattern could be used.

```
target bannerImages: 'item *:image in Banner';
```

Currently, the pattern expressions, though simple, are strong enough to hook into the presentation descriptions for adaptation purposes; however, the pattern matcher can be customized to accept more sophisticated patters.

The *adapt* statements (<ADP-ADAPT>) are used inside an adaptation module to express how the targets are adapted. The target of an adaptation statement is specified after the *adapt* keyword after which a list of parameters (not currently used) follows. Then, a block of statements will affect the targets. Although any sentence is syntactically allowed to be used inside the block, it is not semantically legal or meaningful to have certain statements. It is presumed that the compiler performs validity checks to enforce this. Rule 38 defines a statement; the <OPERATION> statements are specifically defined to be used within adaptation blocks. The operation statements along with the conditional statements (<IF-STATEMENT>, rule 43) define conditional or non-conditional adaptation statements. As rule 42 shows, an operation starts with a '!' character, and has a name and a list of parameters. Table 2 lists some of the operations that can be used to affect the targets in the adaptation packages. These operations are easily extendible. The adaptation properties are defined for later extension of the adaptation model.

Concepts (<CONCEPT>) are composed of a set of attributes that can represent a domain concept, or rather, a piece of information structure. Relationship declarations indicate what relationships exist between the concepts. The actual relationship definitions are defined through the rules 28 to 32. The relationship definition follows the predicate logic ideas in an object-based manner. The relationship body consists of a set of *clauses* that are connected with logical *or*. If any of the clauses

Table 2
Adaptation Operations

Operation Name	Parameter(s)	Description
show		make the item visible
placeTop		place at the top of the layout
placeBottom		place at the bottom of the layout
placeRightmost		place on the rightmost part
placeLeftmost		place on the leftmost part
placeAbove	x: item	place above x
placeBelow	x: item	place below x
placeLeft	x: item	place on the left of x
placeRight	x: item	place on the right of x
changeColor	x: string	change the color to x
changeBGColor	x: string	change the background color to x
changeScale	x: string	change the scale to x
changeFontFace	x: string	change the font to x
scale	x:realNumber, y:realNumber	scale with x, and y factors
changeBorder	x: string	change the border to x
makeBold		make bold
makeUnderlined		make underlined
makeItalic		make italic
makeStrong		make strong
makeEmphasized		make emphasized

evaluates to be true, then the relationship holds. Each clause is a set of predicates connected with logical *and* operator. Predicates can be predefined, such as comparisons, or can defined by the designer based on the domain's requirements.

A statement (rule 38) can be an assignment (<ASSIGNMENT>), an expression (<EXPR>), an operation (<OPERATION>), a fact definition (<FACT>), an instantiation (<INSTANTIATION>), a query (<QUERY>), or a conditional statement (<IF-STATEMENT>). An assignment simply evaluates an expression and assigns the result to an attribute. The expression is produced from rule 49 and its definition is quite similar to the traditional expressions in common programming languages.

A *fact* inserts a record of information, into the system's knowledge base, about a relationship between two concepts. For instance, the prerequisite relationship between two course concepts x and y can be asserted into the system by *.prerequisite(*x*,* y*)*, given that *prerequisite* relationship is declared to be a valid relationship between x and y concept types. On the other hand, queries inquire about existing knowledge through calling the relationships and passing parameters to them. If the relationship holds, then a value of true is returned.

The other production rules are self-explanatory and general in the sense that most of the modern languages support them in one way or the other. Especially, expressions are very common elements in programming languages, and are not elaborated here.

4.4 AWL Adaptive Web System Development Methodology

Although software development process is much more crucial when it comes to general purpose languages (compared to domain-specific ones), it is a good practice that the best development process be outlined by the designer of the language whenever a new language is proposed. AWL software development process is proposed as follows:

1) Analyze domain (this is the application domain, such as e-News, e-Tailer, etc.) and abstract its common notions into *concepts* and the *relationships* between them.
2) Design the presentation fragments in a hierarchical and reusable fashion, based on the functionality that the application is supposed to offer. The presentations should be designed with adaptation in mind. That is, the designer should provide enough flexibility in a presentation to make it adaptable.
3) Package frequently-used statements into functions.
4) Extract frequently-used information and implement them as *extended entities* to make the programming easier for authors.
5) Extract the related user's characteristics and implement them as a user model. Express the dynamic behaviour of the user model through *events*.

6) List all the adaptation instances that the system should provide. This list should be processed to extract related adaptations and abstract them into adaptation modules.

4.5 AWL Design Perspectives

AWL language has been designed to support the development of adaptive Web systems. The languages that support Web development, such as PHP and Perl, are too general-purpose to be regarded as a domain-specific language. They do not offer any specific model, abstraction, or construct to support expressing the adaptiveness of Web applications. AWL is designed to offer more focused support for expressing the adaptivity of Web systems, compromising the generality of them. AWL allows separation of concerns through provisioning different models. The presentation model of an application is independent of its adaptation model; if the adaptation model is removed from an application description, the result would be a non-adaptive version of the application.

AWL can be viewed from two perspective. It can be regarded as a high-level description language that aims at simplifying application development using the adaptive Web systems framework which was developed in IAS group. From this perspective, the design of AWL is the process of providing high-level interfaces and constructs to the framework's low-level facilities. The result will be a framework-dependant language with a limited coverage of the adaptive Web systems.

AWL can also be regarded as a specification language that provides abstractions and constructs that allow the authors to describe the different aspects of adaptive Web systems. From this perspective, AWL is the result of a reuse-based software engineering process that starts with domain analysis and framework implementation. Therefore, the language structure and constructs originate from the notions and terminologies that exist in the target domain. This approach leads to a language that is independent from a specific framework.

In this work, we emphasize on the second perspective, however, in compiler implementation phase, the first perspective is dominant.

AWL is a declarative language in the sense that, it allows the authors specify what an application is composed of. The presentation model of the application is specified in a declarative fashion; the final presentation might be different depending on the request context.

5 AWL Compiler Implementation

We explain the development of AWL Compiler in this section. AWL Compiler is a compiler that translates programs written in AWL into adaptive Web systems framework components.

5.1 The Target Framework

Adaptive Web Systems Framework [30] has been developed by the IAS research group, in the Faculty of Computer Science, at the University of New Brunswick, to provide the necessary components, protocols, and basis for building adaptive Web systems. The framework has a *Synthesis Engine* that accepts a Web site description in a RDF [31] file and synthesizes Web pages for each request. Also, a *User Front-End* component is responsible for receiving HTTP requests and transforming them into synthesis requests. The framework supports domain-specific processing through *Conceptual Tasks*, which are independent components that run as a process, possibly on different machines, and provide the synthesis engine with requested information. There are generic *Conceptual Tasks* in the framework that can be used for applications in any domain. For instance, *User Profile Manager* accepts an RDF description of a user model, and creates the necessary back-end database to support the user model handling at run-time. There is, also, a *Device Profile Manager* that keeps track of different connecting devices' characteristics and provides requested information for synthesis engine.

AWL can be translated into components, models, and artifacts of the adaptive Web framework. In fact, the AWL Compiler has been developed to support the framework with a high-level interface. Some of the advantages of AWL Compiler are as follows:

- AWL Compiler hides the unnecessary details of the framework from the author; hence, the author would not be concerned with how different framework components work together, or how a protocol works; instead, (s)he can just focus on the problem itself and use the high-level constructs and abstractions of AWL to express the solution.
- The possibility of misusing the framework's features is removed by using AWL Compiler, since AWL Compiler is in charge of using the framework's features.
- When AWL Compiler is used to develop an application, the description of the application is an AWL program. Therefore, if in any circumstances the framework internals change, it would not affect the developed application; only AWL Compiler for the framework needs to be changed.
- Because of the high-level constructs of AWL, system development is much faster, simpler, and more productive.

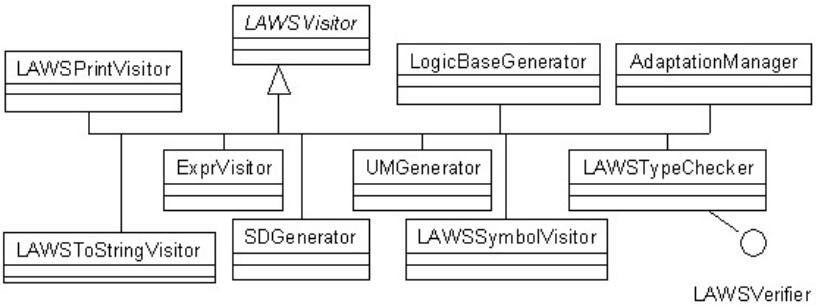


Fig. 4. Several compiler components are defined as visitors

- Reusability of programs and artifacts that are developed in AWL allows the authors to save a substantial amount of time when developing new adaptive Web systems.
- Systems developed using AWL Compiler are easily modifiable.
- AWL Compiler offers various verification facilities that can be used to verify developed systems.

The following section takes a closer look into the design of AWL Compiler.

5.2 Compiler Design and Implementation

AWL Compiler includes the traditional compiler components, such as lexical analyzer, parser, semantic analyzer, and code generator. In the following subsections, different AWL Compiler's components are explained.

5.2.1 Lexical Analysis and Parsing

AWL Compiler uses JavaCC [32] to generate the lexical analyzer and the parser. JavaCC accepts a special notation as a language syntax specifier and generates a Java class that accepts programs in the specified language. The EBNF syntax of AWL is converted to the input format that JavaCC accepts. However, a parser that only accepts input programs is of no use. The ultimate goal of the parser is to generate the abstract syntax tree of input programs. As explained in the previous section, several classes are designed to represent the semantics of the language production rules. These classes are used to enhance the AWL Compiler's parser to generate an abstract tree representing the input program. Later phases of the compiler use the generated abstract tree as input. The generated tree only has useful information about the program and excludes the syntactic sugar (elements of the language that added for readability and beauty of programs).

After the abstract tree is generated, it has to be semantically processed and verified. Most of AWL Compiler's components have been developed following the *visitor*

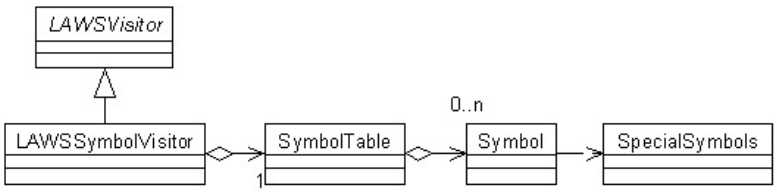


Fig. 5. LAWSSymbolVisitor and supporting classes

design pattern [33]; each component that processes the tree is implemented as a visitor that accepts the abstract tree as input. Figure 4 shows some of the visitor classes implemented in AWL Compiler. All of the visitors extend the abstract *LAWSVisitor* class, which provides methods for visiting all the semantic objects generated by the parser. The simplest visitor is *LAWSPrintVisitor*, which simply traverses the whole tree and prints the information inside its nodes. The other visitors will be explained in the following sections.

5.2.2 Symbol Table Visitor

The symbol table keeps information about the program entities that are not part of the keyword set of the language, such as item and attribute names. Figure 5 shows the *LAWSSymbolVisitor* class, which is derived from the *LAWSVisitor* class. The elements of the symbol table are of type *Symbol*, which has a symbol table embedded. This originates form the structure of AWL that allows some entities to be defined inside other entities. Each symbol object has information about the symbol's name, type, address, and semantic object (generated by the parser). *LAWSSymbolVisitor*, provides these information, for other compiler components, to be used in later phases.

5.2.3 Register Management

Synthesis engine has a bank of registers, which is used for storing various information about the current user request and the current Web application. These registers are the only memory storage that compiler-generated systems can use to store information. Registers are mostly used for storing the results of information queries from *Conceptual Tasks* as well as intermediate results of expression evaluation. The attributes of each presentation fragment are assigned a location in the register bank and the address is assigned to the corresponding symbol in the symbol table.

5.2.4 Adaptation Weaver

Since the adaptation modules are expressed separately from the presentation descriptions, it is the compiler's responsibility to weave the adaptation into the presentation descriptions. This is accomplished by the *Adaptation Manager* (Figure 6).

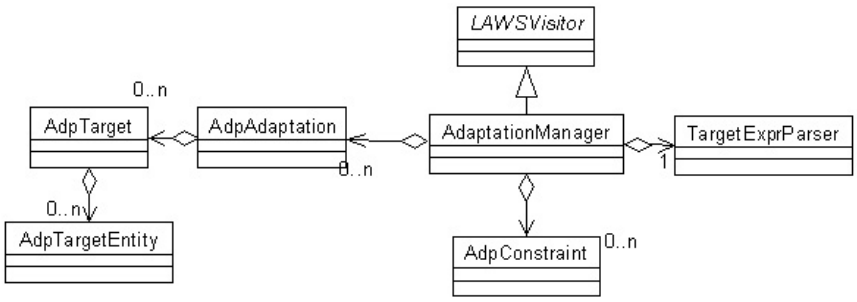


Fig. 6. Adaptation Weaver and the helper classes

```
<sdv:CTPut rdf:ID="CVP1cmd1">
  <!--  User Profile Manager Task ID -->
  <sdv:cTaskID>5555</sdv:cTaskID>
  <!--  Request: checkValue  -->
  <sdv:reqID>23</sdv:reqID>
  <rdf:value>
    <rdf:Seq>
      <rdf:li rdf:resource="#SEReg1." />
      <rdf:li>newsID</rdf:li>
      <rdf:li rdf:resource="#SEReg37." />
    </rdf:Seq>
  </rdf:value>
</sdv:CTPut>
```

Fig. 7. Hand-coded *Conceptual Task* call in RDF description

Adaptation Manager uses a parser class to interpret the target pattern string used to specify adaptation targets. Then, the target item's semantic object is retrieved through the symbol table. For each adaptation, affecting an item, a constraint object is created, which encapsulates the necessary instructions and conditions that enforce the adaptation of the item.

5.2.5 Service Manager

Conceptual Tasks provide several services that can be used by applications using the framework. Figure 7 shows part of the generated code from a Web site description that calls a service form the *User Profile Manager (UPM) Conceptual Task* to check if a news item has been read by the user. As the picture shows, it is not very easy to make service calls; especially when service composition is needed. In AWL, services can be called similar to a function call. For instance, for the service call in Figure 7, the author can write:

```
upm.checkValue(userID, 'newsID', news);
```

where, *userID* is a keyword; 'newsID' is the attribute name whose value is being checked; and news is an attribute defined somewhere and is supposed to hold the news ID. AWL Compiler translates this code to the low-level code that is understandable by *synthesis engine*.

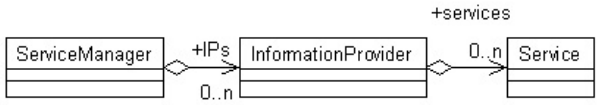


Fig. 8. Service Manager and the supporting classes

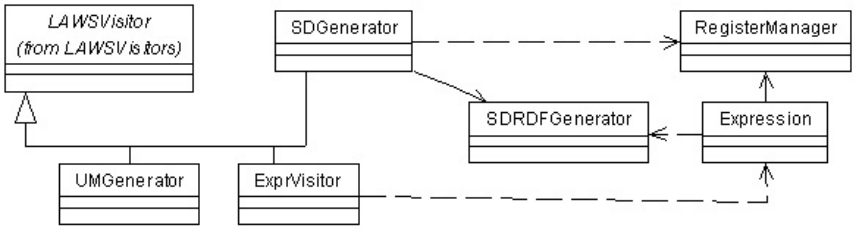


Fig. 9. Code generators class hierarchy

In order to translate the service calls in AWL programs to low-level RDF code, the compiler has to know the existing *Conceptual Tasks* and the services they offer as well as their parameters and return types. Service Manager (Figure 8) manages the conceptual tasks and their services and provides necessary information for the compiler to translate service calls in programs to low-level RDF description.

5.2.6 Application Description Generator

Figure 9 shows the heart of the compiler, *i.e*, *SDGenerator*. This class is responsible for generating the final site description using the other components of the compiler such as register manager and expression evaluation code generator. *SDRDFGenerator* offers methods for creating RDF objects for framework entities. For expression code generation, each expression in the abstract tree is passed to *ExprVisitor*, which, in turn, generates an expression tree whose nodes are objects of type *Expression*. *Expression* class has a method called *generatePostfixEval*, which processes the expression tree and generates RDF code that will evaluate the expression at runtime. As Figure 9 shows, *UMGenerator* is another visitor class that generates the user model in RDF format.

5.2.7 Logic-Base Generator

The concepts and relationships that are defined in an AWL program best fit a logic-based paradigm of programming. This comes from the fact that, concepts and relationships should be defined for later queries and inference; that is, similar to the logic programming language paradigm, after providing rules and facts in a domain, it is desirable to inquire about unknown predicates. AWL Compiler translates concepts and relationships to Prolog predicates. These predicates, then, are read by a framework's component called *Concept Manager*, which is a logic base that provides run-time services for updating domain information modeled as concepts and

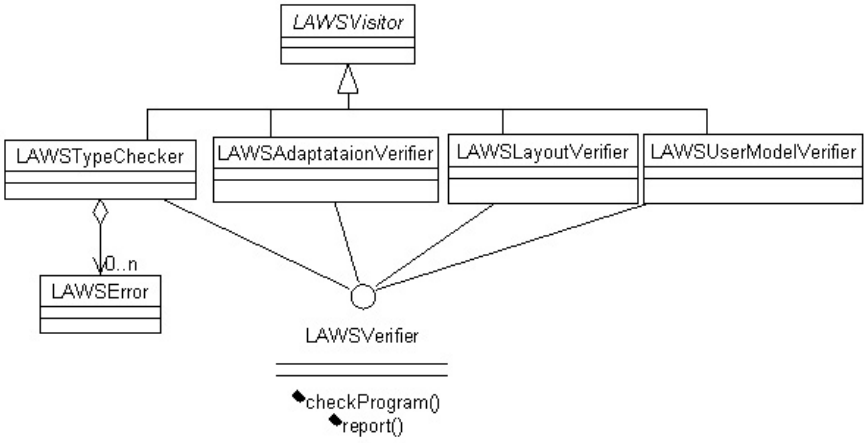


Fig. 10. Class hierarchy of program verifiers

relationships; besides, it provides facilities for querying the logic base.

5.2.8 Program Error-Checking and Verification

Figure 10 shows the hierarchy of verifier classes that accept the semantic object hierarchy (created by the parser) and verify it against language semantic rules. For instance, *LAWSTypeChecker* will check the expressions and statements to find any type inconsistencies. Once it finds such inconsistencies, it creates a *LAWSError* object and in the end, reports all the errors.

Other verifiers check programs for their soundness. *LAWSLayoutVerifier* builds a graph of all the placement relationship between items of a presentation and through executing an algorithm, finds any layout inconsistencies between the items. For instance, if item x is said to be *above* item y, and item y is said to be at *top*, this is a violation and will be reported by the verifier. The algorithms and further information can be found in [6].

6 Example Application

In this section an example system, which is developed in AWL, is presented. First, the application functionalities and features are explained. Then, the program modules of the application are presented.

6.1 PENS: Personalized E-News System

Personalized Electronic News System (PENS) [34] is an adaptive Web system that provides the user with the most relevant news, taking into account the user's brows-

Fig. 11. PENS front page

ing history, interests, and location. In addition, PENS takes the target device's characteristics (limitations and features) into account in order to deliver the most suitable information presentation to the client.

The front page of PENS, which is what users see when they access the system, is shown in Figure 11. It has a banner, at the top, and a main section below the banner. The banner is adaptive to the location of the user. That is, if the user is connecting from UNB's Saint John campus, then the banner will include the "Saint John" image; if the user is connecting from Fredericton campus, then the "Fredericton" image is shown in the banner. On the left side of the main section, a list of different news categories is provided so that the user can view the most recent news in any of the categories. On the right side is the section presenting the most recent news items. This section, in turn, is divided into two sub-sections: the upper sub-section, "top news" , which shows three news items with their abstracts, and the lower sub-section, "more topics", which shows just the titles of four more news items.

The "new" icon beside a news item indicates that the item is new since the last visit of the user. This is a navigation adaptation through link annotation. Each news title is linked to a page that presents the full news body. Figure 12 shows a full news body page. In the full news body, in addition to news correspondent and full text, there is a section that provides a link to the most related news item to the current news. This is considered as navigation adaptation through link suggestion.

Fig. 12. The full news page

Fig. 13. The front page after a full news visit

When the user visits the front page, after reading a news item, (s)he will not see the abstract part of the visited news item. This is done based on the assumption that the user is most likely not interested in the abstract of the news items that (s)he has already read. This is content adaptation through conditional fragments. Figure 13 shows how the front page may look, after the user reads the full news page. There is also another adaptation that takes place in PENS. The news items in front page are sorted according to their publish date. That is, the most recent news will come first from the top. However, if the user is connected from Fredericton campus, and the top news is related to Saint John campus, while the second top news is related to Fredericton campus, then the second top news item is moved up above the otherwise top news item. The same adaptation applies when the user connect from Saint John campus. This adaptation only takes place for the "top news" news items.

7 Conclusions and Future Work

7.1 Conclusions

In this work, we proposed AWL as a new language for adaptive Web systems development. Various production issues and shortcomings in the adaptive Web framework inspired us to look for solutions that not only address those issues, but also bring in extra features that enhance the quality of developed software.

AWL can be viewed from two perspectives: a high-level description language for our framework, and a domain-specific language for adaptive Web systems. The

contributions of this work can also be viewed from each of these perspectives.

As a high-level description language for the adaptive Web framework, AWL aims at simplifying application development using the framework which was developed in IAS group. From this perspective, the design of AWL is the process of providing high-level interfaces and constructs to the framework's low-level facilities.

As a domain-specific language for adaptive Web systems, AWL provides abstractions and constructs for authors to model applications in the adaptive Web domain. From this perspective, AWL is the result of the adaptive Web domain engineering process; users of AWL do not need to know the details of the domain implementation. They only need to know the application engineering process that AWL suggests. Therefore, the commonalities of adaptive Web domain are realized and implemented once and the AWL compiler will make use of them. Authors (programmers) just need to express the variations through the AWL.

In Section 4, based on the analysis of adaptive Web systems, the language requirements were explained, and then the EBNF syntax was presented. The semantics of the language was also explained using UML class diagrams. Then, a development process was suggested to be used when developing in AWL.

AWL Compiler development was discussed in Section 5. AWL Compiler was specifically developed to translate programs in AWL into the adaptive Web framework's components and models. It also provides facilities for verification of programs.

PENS, personalized e-News system, was presented as the proof-of-concept application developed to prove the functionalities of the adaptive Web framework as well as AWL language.

7.2 Future Work

Potential future work is suggested as follows:

- AWL needs to go through an extensive review, evaluation, and revision. For instance, one of the biggest challenges in designing domain-specific languages is the decision on the generality of the language. It is of great importance to find out if AWL is too general purpose (within the adaptive Web domain), or too specific, hence limited, to express some features.
- There are many domains, within the adaptive Web systems, in which we have not developed any application using AWL. It is important to use AWL to develop applications in other domains; the results of these developments can greatly contribute to the AWL evolution. For instance, it should be very interesting to know how AWL facilitates adaptive e-Book or adaptive e-Tailer development.
- AWL is a two-level language, meaning that, it is designed to be used to develop

both domain libraries and applications. It is very useful to select a set of candidate domains and develop domain libraries for them using AWL. These libraries, then, would be provided along with AWL Compiler, to help authors easily develop applications for those domains. As mentioned in Section 6, most of the development time in AWL is put for developing artifacts that are reusable for an application domain and will not be reinvented in case a domain library is used.

- Visual tools can increase the productivity of a compiler. It is a valuable effort to build an IDE (Integrated Development Environment) to harness the power and facilitate the use of AWL Compiler.

References

[1] M. Kilfoil, A. Ghorbani, W. Xing, Z. Lei, J. Lu, J. Zhang, X. Xu, Toward an adaptive web: The state of the art and science, in: the 1st Annual Conference on Communication Networks and Services Research (CNSR 2003), 2003, pp. 119–130.

[2] H. Wu, P. D. Bra, A. T. M. Aerts, G.-J. Houben, Adaptation control in adaptive hypermedia systems, in: Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer-Verlag, 2000, pp. 250–259.

[3] P. D. Bra, A. Aerts, B. Rousseau, Concept relationship types for AHA! 2.0, in: E-Learn'02, Montreal, Canada, AACE, 2002, pp. 1386–1389, http://www.win.tue.nl/ debra/elearn2002/brendan.pdf.

[4] J. A. Macas, P. Castells, Adaptive hypermedia presentation modeling for domain ontologies, in: 10th International Conference on Human-Computer Interaction (HCII 2001), New Orleans, Louisiana, 2001, pp. 710–714, http://www.ii.uam.es/ castells/publications/hcii01.pdf.

[5] N. P. de Koch, Software engineering for adaptive hypermedia systems, Ph.D. thesis, Ludwig-Maximilians-Universitt Mnchen (2000).

[6] S. Hossein Sadat K. M., Ali A. Ghorbani, A presentation description language for adaptive web systems, in: Communicaiton Networks and Services Research (CNSR 2005), IEEE, Halifax, Canada, 2005, pp. 169–175.

[7] P. D. Bra, N. Stash, AHA! adaptive hypermedia for all, in: SANE 2002 Conference, Maastricht, 2002, pp. 411–412.
URL `citeseer.nj.nec.com/debra02aha.html`

[8] G. Weber, H.-C. Kuhl, S. Weibelzahl, Developing adaptive internet based courses with the authoring system netcoach, in: Revised Papers from the nternational Workshops OHS-7, SC-3, and AH-3 on Hypermedia: Openness, Structural Awareness, and Adaptivity, Springer-Verlag, 2002, pp. 226–238.

[9] P. Brusilovsky, D. W. Cooper, ADAPTS: Adaptive hypermedia for a web-based performance support system, in: Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999, pp. 41–47.

[10] J. E. Peter Brusilovsky, E. Schwarz, Web-based education for all: a tool for developing adaptive courseware, in: Computer Networks and ISDN Systems, Vol. 30, Nos. 1–7, 1998, pp. 291–300.
URL
http://www7.scu.edu.au/programme/fullpapers/1893/com1893.htm

[11] M. Cannataro, A. Cuzzocrea, A. Pugliese, XAHM: an adaptive hypermedia model based on XML, in: Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM Press, 2002, pp. 627–634.

[12] N. Koch, Software engineering for adaptive hypermedia systems:reference model, modeling techniques and development process, Ph.D. thesis, Ludwig-Maximilians-University of Munich (December 2000).

[13] P. D. B. Hongjing Wu, Geert-Jan Houben, Aham: A dexter-based reference model for adaptive hypermedia, in: Proceedings od the 10th ACM Conference on Hypertext and Hypermedia, Darmstadt, Germany, 1999, pp. 147–156.
URL http://wwwis.win.tue.nl/ hongjing/pub/ht99.ps

[14] P. D. Bra, Design issues in adaptive web-site development, in: Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999, pp. 29–39.

[15] P. D. Bra, N. Stash, B. D. Lange, AHA! adding adaptive behavior to websites, in: Proceedings of the NLUUG Conference, Ede, The Netherlands, 2003, pp. 21–23.
URL citeseer.nj.nec.com/585332.html

[16] P. D. Bra, A. Aerts, B. Berden, B. D. Lange, B. Rousseau, T. Santic, D. Smits, N. Stash, AHA! the adaptive hypermedia architecture, in: Proceedings of the ACM Hypertext Conference, ottingham, UK, 2003, pp. 81–84.
URL http://wwwis.win.tue.nl/ debra/ht03/pp401-debra.pdf

[17] L. Ardissono, A. Goy, R. Meo, G. Petrone, L. Console, L. Lesmo, C. Simone, P. Torasso, A configurable system for the construction of adaptive virtual stores, World Wide Web 2 (3) (1999) 143–159.
URL citeseer.nj.nec.com/209232.html

[18] M. Perkowitz, O. Etzioni, Adaptive web sites: Automatically synthesizing web pages, in: AAAI/IAAI, 1998, pp. 727–732.
URL citeseer.nj.nec.com/perkowitz98adaptive.html

[19] L. Ardissono, L. Console, I. Torre, An adaptive system for the personalized access to news, AI Commun. 14 (3) (2001) 129–147.

[20] L. Ardissono, A. Goy, G. Petrone, M. Segnan, L. Console, L. Lesmo, C. Simone, P. Torasso, Agent technologies for the development of adaptive web stores, in: Agent Mediated Electronic Commerce, The European AgentLink Perspective., Vol. vol. 1991, Springer-Verlag, London, UK, 2001, pp. 194–213.

[21] L. Ardissono, C. Barbero, A. Goy, G. Petrone, Adaptive web stores, in: Agents'99 Workshop: Agents for Electronic Commerce and Managing the Internet-Enabled Supply-Chain, 1999, pp. 9–13.
URL citeseer.nj.nec.com/218285.html

[22] M. Milosavljevic, J. Oberlander, Dynamic hypertext catalogues: Helping users to help themselves, in: Proceedings of the 9th ACM Conference on Hypertext and Hypermedia (HT'98), Pittsburgh, PA, USA, 1998, pp. 20–24.
URL citeseer.nj.nec.com/milosavljevic98dynamic.html

[23] A. van Deursen, P. Klint, J. Visser, Domain-specific languages: an annotated bibliography, SIGPLAN Not. 35 (6) (2000) 26–36.

[24] H. Mili, A. Mili, S. Yacoub, E. Addy, Reuse-Based Software Engineering, Techniques, Organization, and Controls, John Wiley & SONS, INC., 2002.

[25] D. Alur, J. Crupi, D. Malks, Core J2EE Patterns: Best Practices and Design Strategies, 1st Edition, Prentice Hall, 2001.

[26] Hossein Sadat, Ali A. Ghorbani, On the evaluation of adaptive web systems, in: to appear in WSS04, The Second International Workshop on Web-based Support Systems in conjunction with AI 2004, Beijing, China, 2004, pp. 127–136.

[27] S. Robert W, Concepts of Programming Languages, 4th Edition, Addison Wesley, 1999.

[28] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, J. Irwin, Aspect-oriented programming, in: ECOOP97 Object-Oriented Programming, 11th European Conference, volume 1241 of Lecture Notes in Computer Science, 1997, pp. 220–242.
URL
http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales -ECOOP97/for -web.pdf

[29] EBNF Syntaxt Specification Standard, EBNF: ISO/IEC 14977 : 1996(E).
URL http://www.cl.cam.ac.uk/ mgk25/iso-14977.pdf

[30] IAS Group, Adaptive web sites (AWS) framework: High-level design document, Tech. Rep. TR03-102, Intelligent and Adaptive Systems Researh Group, Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada (December 2003).

[31] W3C, Resource description framework (RDF).
URL http://www.w3.org/RDF/

[32] Sun Microsystems, Java Compiler Compiler (JavaCC), the Java Parser Generator.
URL https://javacc.dev.java.net/

[33] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1st Edition, Addison Wesley, 1995.

[34] M. Nadjarbashi-Noghani, J. Zhang, S. Hossein Sadat K. M., Ali A. Ghorbani, PENS: A personalized electronic news system, in: Communicaiton Networks and Services Research (CNSR 2005), IEEE, Halifax, Canada, 2005, pp. 31–38.

APPENDIX: PERSONALIZED ELECTRONIC NEWS SYSTEM (PENS)

A PENS' Application Code in AWL

The following sections present the different models of PENS system, coded in AWL.

A.1 PENS' Presentation Model

The presentation model is an important model for an application since without it nothing can be presented to the user. Also, a program in AWL can be translated successfully with only a presentation description. This originates from the idea that an adaptive Web application should be described in a way that if its adaptation model is removed, then the rest of the application should work similar to a non-adaptive version of the application. Since AWL has been designed based on the "separation of concerns" principle, the separation of presentation and adaptation aspects are automatically supported.

```
main fragment frontPage {

        header:
                title 'News@UNB Compiler Generated Application';
                identified 'card1';
                entitled 'News@UNB!';
        header;

        item banner: Banner('newsAtUNB.gif', 'unbLogo.gif', 'fred.gif', 'sntj.gif', 'banner.jpg')
        {
                top;
        }

        item mainSection: Section
        {
                border = 'InFrame';
                top;
        }
}
```

Fig. A.1. The fragment definition of the front page of PENS

```
fragment Banner {

        attribute im1: String, im2: String, im3: String,
                im4: String, im5: String;

        item img1: Image(im1)
        {
                width = 323; height = 32;
        }
        item img3: FredSj(im3, im4) {}
        item img2: Image(im2)
        {
                width = 162; height = 76;
        }
        item img5: Image(im5)
        {
                border = 'FloorNCeil';
                width = 644; height = 80;
        }

        init: i1:String, i2:String,i3:String, i4:String, i5: String
        {
                im1 = i1; im2 = i2; im3 = i3;
                im4 = i4; im5 = i5;
        }
}
```

Fig. A.2. *Banner* fragment definition

It should be noted that presentation abstraction and design highly depend on the designer's concerns. For instance, one might consider reusability as a very important quality factor when designing presentation model. Others might aim at a smaller presentation model.

Figure A.1 shows the front page presentation description. Since AWL is a domain-specific language designed for adaptive Web systems, it is very simple to find out what a presentation specification means. The header section is used only in fragments that are to be rendered as Web page. It is the place in which the author can specify title and meta information for webpages. The *main* modifier indicates that this presentation fragment is the main presentation, hence the front page, of the application. The front page is composed of two fragments: a banner, which is a fragment of the type Banner with parameters that specify the image names in the banner, and a main section that is a fragment of type Section. The banner is set to be at the top of the presentation, and the *mainSection* is set to have a frame border.

```
fragment Section {

        item categories : CategorySection
        {
                bgcolor = '#E0EEEE';
                valign = 'top';
                left news;
        }
        item news: NewsBoard {}
}
```

Fig. A.3. *Section* fragment definition

```
fragment NewsBoard {

        item firstSection: DetailedNews {}
        item secondSection: ShortNews {}
}
```

Fig. A.4. Fragment definition of a news board

```
fragment FredSj {

        attribute im1: String, im2: String;

        item img1: Image(im1)
        {
                width = 150; height = 19;
        }
        item img2: Image(im2)
        {
                width = 150; height = 19;
        }

        init: i1:String, i2:String
        {
                im1 = i1; im2 = i2;
        }
}
```

Fig. A.5. Fragment definition of FredSJ item type

Figure A.2 shows the definition of fragment *Banner*. *Banner* is composed of three images and a custom fragment *FredSj*. As Figure A.5 shows, *FredSj* fragment is composed of two images. The *init* section of each fragment is used to initialize its attributes. The parameters of *init* block hold the values sent to the fragment in a type instantiation.

The *Section* fragment (Figure A.3) is composed of a *CategorySection* and a *NewsBorad* (Figure A.4). The background color of the category section is set to be different than the news section. In addition, it is positioned on the left of the news board. Figures A.6, A.8,and A.7 show some other parts of the PENS presentation specification.

A.2 PENS' User Model

The user model for PENS has been designed in a special fashion. First, all the attributes and structures that are needed to define a user model in e-News domain are defined and abstracted in a user model package (Figure A.10). Then, this base package can be used for initializations, extensions, and dynamic behaviour specifications. In PENS though, there in not much of an extension; however, the user

```
fragment DetailedNews {

        attribute newsIDs : list[3] of String;

        item title : Text('TOP NEWS')
        {
                largefontsize; italic; bold; underlined;
                color = 'maroon';
                textScale = '100%';
                fontFace = 'helvetica';
        }

        verticalBox(3)
        item newsitems[$]: NewsItem(newsIDs[$]) {}

        init:
        {
                newsIDs[0] = call NewsFeeder.getNewsID(1);
                newsIDs[1] = call NewsFeeder.getNewsID(2);
                newsIDs[2] = call NewsFeeder.getNewsID(3);
        }
}
```

Fig. A.6. Fragment definition of *top news* section

```
fragment NewsItem {

        attribute newsID: String;
        attribute title: String;
        attribute abstract: String;

        item headline: Text (title) {
                linksTo fullNews (newsID);
        }

        item icon : Image('new.gif') {
                right headline;
        }

        item abstract: Text (abstract) {
                length = 150;
        }

        init:ne: String {
                newsID = ne;
                title = call NewsFeeder.getHeaderByID(newsID);
                abstract = call NewsFeeder.getBodyByID(newsID);
        }
}
```

Fig. A.7. Fragment definition of a news item

```
fragment fullNews  {

        attribute newsID: String;
        attribute relatedNewsID: String;
        attribute hl: String;

        header:
                title hl;
                identified 'card1';
                entitled 'News@UNB - University of New Brunswick';
        header;


        item ban: Banner('newsAtUNB.gif', 'unbLogo.gif', 'fred.gif', 'sntj.gif', 'banner.jpg') {
                top;
        }

        item headline: Text (hl) {}
        item info: Text (call NewsFeeder.getCorrespondent (newsID)) {}
        item full: Text (call NewsFeeder.getBodyByID(newsID)) {}

        item relatedText: Text ('Related News') {
                largefontsize; italic; bold;
                color = 'maroon';
                textScale = '100%';
                fontFace = 'helvetica';
        }
        item related: Text (call NewsFeeder.getHeaderByID(relatedNewsID))    {
                linksTo fullNews (relatedNewsID);
        }
        item return: Text ('return to main page!') {
                italic;
                linksTo frontPage;
        }

        init: ns: String {
                newsID = ns;
                relatedNewsID = call AssociationMiner.recommend(userid, newsID, 1);
                hl = call NewsFeeder.getHeaderByID(newsID);
        }
}
```

Fig. A.8. Fragment definition of the full news page

```
adaptation BannerAdaptations {

        target fred : 'item img1 in FredSj';
        target sj : 'item img2 in FredSj';

        adapt fred
        {
                if call iplocator.locateip(ClientIP) == 1
                {
                        !show;
                }
        }

        adapt sj
        {
                if call iplocator.locateip(ClientIP) == 0
                {
                        !show;
                }
        }
}
```

Fig. A.9. Banner adaptation based on user location

model updating behaviour is specified through an *event* construct (Figure A.11).

```
usermodel enewUserModel {
        attribute       test: realNumber,
                        userID: String,
                        password: String,
                        dgInfor: DGModel,
                        groupInfor: GroupInfor,
                        interestedTopics: InterestedTopics,
                        visitedNews: VisitedNews,
                        domainExperties: DomainExperties,
                        detailedLevel: DetailedLevel,
                        advertisements: Advertisements;

        define DGModel {
                attribute age: realNumber,
                        gender: alternative{'male', 'female'},
                        educationLevel: alternative {'high school', 'bachelor', 'master', 'phd'},
                        educationField: alternative {'Computer Science', 'Business Administration',
                                                'Electric Engineering', 'Chemical Engineering'
                        occupation: alternative {'programmer', 'Manager', 'CEO'},
                        jobField: alternative {'IT', 'Business', 'Engineering', 'Chemical Engineering'},
                        englishLevel: alternative {'basic', 'Medium', 'Advance'},
                        email: String
                           }
                .
                .
                .
}
```

Fig. A.10. Part of a user model for e-News domain

```
main usermodel newusermodel extends enewUserModel
{
        events:
        on visit (fullNews)
        {
                call upm.addValue(userid, 'newsID', newsID, 'sessionID',
                                sessionID, 'newsCat', 'ACADEMIC');
        }
}
```

Fig. A.11. The PENS' user model derived from the e-News user model

```
adaptation UserbasedAdaptations {

        target abstract : 'item abstract in NewsItem';
        target icon : 'item icon in NewsItem';

        adapt abstract
        {
                if call upm.checkValue(userID, 'newsID', newsID) == 0
                {
                        !show;
                }
        }

        adapt icon
        {
                if(     (call upm.getLastSessionDate(userID, 'sessionID', sessionID)
                        <
                        call NewsFeeder.getNewsDate(newsID))
                  and
                        call upm.checkValue(userID, 'newsID', newsID) == 0)
                {
                        !show;
                }
        }
}
```

Fig. A.12. News abstract removal adaptation

```
adaptation TopNewsAdaptations1 {

        target news2 : 'item newsitems[1] in DetailedNews' ;

        adapt news2
        {
                if (
                    (   call iplocator.locateip(ClientIP) == 1
                     and
                        call NewsFeeder.getLocationByID(parent.newsIDs[0]) == 'saintjohn'
                     and
                        call NewsFeeder.getLocationByID(parent.newsIDs[1]) == 'fredericton'
                    )
                or
                    (   call iplocator.locateip(ClientIP) == 0
                     and
                        call NewsFeeder.getLocationByID(parent.newsIDs[0]) == 'fredericton'
                     and
                        call NewsFeeder.getLocationByID(parent.newsIDs[1]) == 'saintjohn'
                    )
                )

                {
                        !placeAbove (newsitems[0]);
                }
        }
}
```

Fig. A.13. Adaptation description for the top news items

A.3 PENS' Adaptation Model

AWL allows designers to abstract adaptation in a flexible way. Adaptation can be defined in several packages. Each package can hold any number of adaptation.

Figure A.9 shows the adaptation package describing banner adaptation based on user location. Two targets are defined: *fred* and *sj*, referring to *img1* and *img2* items, respectively, in *FredSj* presentation. The adaptation statements then describe how these targets adapt. In case the user location is 1, the *fred* target will be shown, and if the location is 0, then the *sj* target is selected (*IPLocator* is a framework component that provides a service through which the location code for a user IP can be acquired).

Figure A.12 shows the adaptation package that affects a news item in the *top news* section. The first *adapt* statement queries the user profile manager (emphupm, which is a framework component responsible for updating the user model) to find out if the user has already visited the news item. If the user has already read the item, then the news abstract is not shown. The *New* icon is also conditional, and might be removed if the item is not new to the user. Figure A.13 describes the adaptation that affects the order of the items in the *top news* section.

It should be noted that the implementation of PENS includes models that could have been implemented in an e-News domain library, in which case, PENS-specific models and code fragments would have been developed, using the domain library, in much faster and more concise fashion.