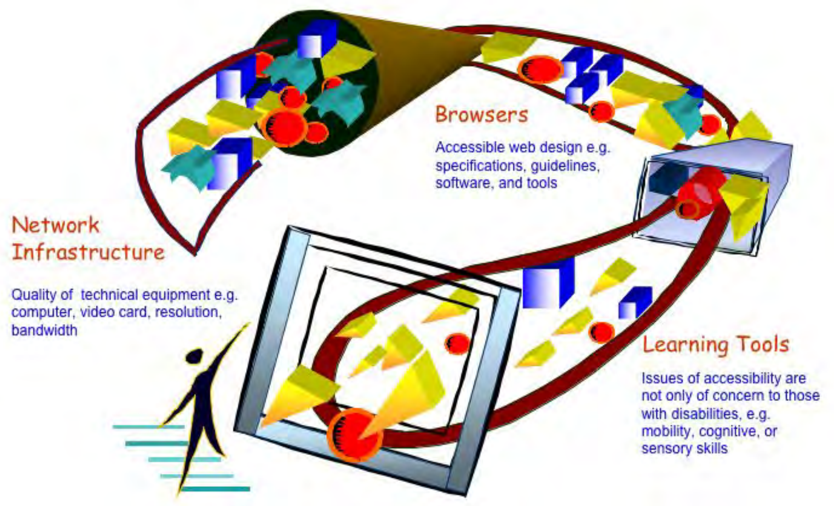
Diversity in an Environment for Accessible Learning (DEAL)

Shohreh Hadian

Camosun College

November 2009

CANADIAN COUNCIL ON LEARNING CCL CCA CONSEIL CANADIEN SUR L'APPRENTISSAGE


Browsers
Accessible web design e.g. specifications, guidelines, software, and tools
Network Infrastructure
Quality of technical equipment e.g. computer, video card, resolution, bandwidth
Learning Tools
Issues of accessibility are not only of concern to those with disabilities, e.g. mobility, cognitive, or sensory skills

This work was funded by a contribution from the Canadian Council on Learning

1. TABLE OF CONTENTS

EXECUTIVE SUMMARY

With the recent increase in the popularity of Web based instruction, it has become important to become aware of the barriers online users face in virtual environments. As technologies emerge to deliver Web-based instruction, it is also important to educate both courseware developers and educators of solutions they can apply to make sure that diversity is one of the parameters that need to be included in the design of **Learning Content Management System** (LCMS). LCMSs are software for delivering and tracking content and they range from systems for managing training/educational records to software for distributing courses over the Internet and offering features for online collaboration. In this context, access with diversity in mind is a key issue for web content developers and users.

The objectives of this project are to address the diversity issues in Web-based teaching and learning. It is understood that in order to make the LCMS tools more accessible to a wider audience, diversity to optimally deliver content needs to be considered. The report is comprised of the main document and three additional appendices: The main document presents an overview of the project with a simple test case implementation of a web site with intensional programming using Markup Macro Processor language (MMP). Appendix 1 outlines the System Design architecture, Appendix 2 describes the details on the Markup Macro Processor Language (MMP) and Appendix 3 contains the Profile Module Summary. The preliminary findings in the current research support the use of MMP in web content development by enabling multi-versioning of web content. The software was designed and implemented as a macro processor and an interpreter to enable the author to insert intensions into the HTML code.

The financial support from the Canadian Council on Learning (CCL) and the CanAssist Research Group at the University of Victoria is acknowledged. The following people have contributed to the research program: Professor Bill Wadge (CSC, UVic), Professor Nigel Livingstone (CANAssist, UVic), Dr. Naz Virjee-Babul (Down Syndrome Research Foundation), and the students Lawrence Lee, Brian Price and Rob Little (Camosun College) and Rich Little (UVic).

2. INTRODUCTION

The Web is being used to teach and to conduct research. The primary users include the universities and institutions for vocational and life-long learning. Thus it is important for developers, educators and students to become aware of the accessibility barriers in web-based environments to ensure that all users are able to participate in this new and rapidly evolving form of education.

The current research takes a small step towards understanding how web-based educational technology can help and enhance teaching and learning taking into account the diversity in learning outcomes. To this end, a new framework has been designed and implemented that facilitates web content authoring, content delivery, and web operation and navigation in order to aid teachers with content and pedagogical development and help learners access information in an optimal way with respect to the particular ability or disability.

Web-based education presents several benefits since the material can be presented in various formats, and it is possible to review supporting material without interrupting the flow of learning. However, the technology implementation has proved far more difficult than anticipated. The accessibility barriers in e-learning and teaching are summarized in Figure 1: (i) web-based learning tools and assistive technologies (ii) browser design issues; and (iii) infrastructure. The primary objective of this study is to research, design, implement and test a tool that provides a repository of learning objects and records a profile of the users, in order to deliver a customized version of the learning object[1] to the user with inclusive design in mind.[1-2]. The intention is to package and deliver content taking into account the diversity in learning subjects and materials encountered by teachers and learners in a classroom setting.

A preliminary study conducted at the University of Toronto in 1998 [3] revealed that the available Web-based teaching tools fail to properly consider accessibility. A subsequent study in 2000 noted considerable improvements in interface design as developers are becoming aware of accessibility barriers [4]. Further studies [5] indicate, however, that web accessibility is about

[1] A Learning Object is a digital and web-based entity or resource that can be aggregated into larger collections of content, can be tagged for searching, are self-contained and can be used and re-used to support learning.

three times better for sighted users than for users with visual impairments. In 2000, Mei and Storey [6] conducted a study on the usability of web-based learning tools. The study revealed that tools seem to enhance learning when they are perceived as being invisible. Furthermore, the study also recommends that the development of web-based learning tools with accessibility in mind is crucial to facilitate the learning process. Also instructors, as the designer of course web sites, play an important role in using web-based learning tools. In a traditional classroom setting, instructors adjust their style of teaching to fit the learner's learning profiles. This is not the case in an online environment. This is the missing link in order to provide an optimal learning experience. By incorporating multi-versioning in content delivery enables the teacher to adapt the delivery of information to individual learning styles.. A more recent thesis by Hadian [7] proposed a definition for accessibility in the context of software engineering and one of the findings pointed to the need to design web based learning tools with diversity in mind, in particular in a college teaching environment.

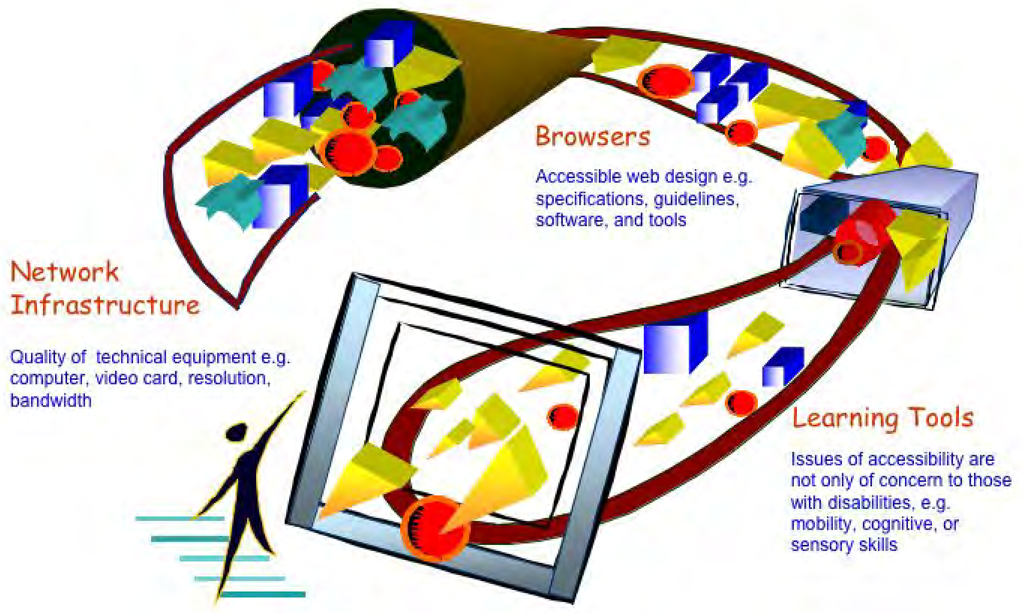

Browsers
Accessible web design e.g. specifications, guidelines, software, and tools
Network Infrastructure
Quality of technical equipment e.g. computer, video card, resolution, bandwidth
Learning Tools
Issues of accessibility are not only of concern to those with disabilities, e.g. mobility, cognitive, or sensory skills

Figure 1 - Accessibility issues in a virtual environment

3. DEFINITION OF ACCESSIBILITY

The primary concern in the development of accessible web-based information lies in the definition of accessibility. On the one hand, web accessibility has been perceived solely as a disability issue. On the other hand, it is an essential component in the development of universal design [8]. Examples of universal design in buildings are the inclusion of ramps and automatic door openers as well as Braille labels on elevator control buttons. Just as wheelchair ramps to buildings have become the norm in architectural design, accessible web design must become a basic consideration in the design of Web-based learning resources. Using universal design principles to create a web resource ensures that all users can access information at a Web site regardless of their abilities, their disabilities, or the limitations of their equipment and software [9].

In order to understand the characteristics related to accessibility, a rationale based on the ESSI-SCOPE software design guidelines was adopted [10], as shown in Figure 2. ESSI-SCOPE is an EU (European Union) funded project that aimed to raise awareness of quality issues in software products.This research has identified a list of desirable features in software design. A subset of these characteristics was selected that impact accessibility and these are depicted in Figure 2. Note that the subset includes only those characteristics that directly affect accessibility.

Definitions:

- **Interoperability** is the attribute that bears on the tool's ability to interact with specified operating systems;
- **adaptability** is the attribute that bears on the opportunity for the tool's adaptation to different specified environments without applying actions or means other than those provided for this purpose for the software considered;
- **operability** is the ability of software that bears on the user's effort for operation and operation control;
- **learnability** is the attribute that bears on the user's effort for learning its application;
- **changeability** is the attribute that bears on the effort needed for modification and environmental change; and

- **fault-tolerance** is the attribute of the software that bear on its ability to maintain a specified level of performance in case of software faults.

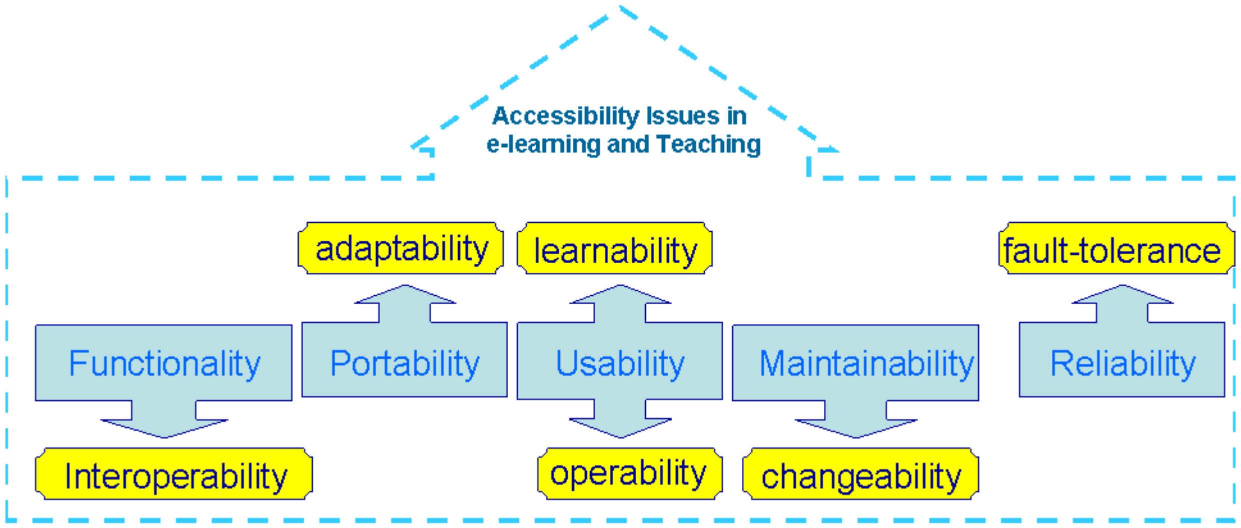


Figure 2 – Definition of Accessibility

4. MOTIVATION

Educators and learners play a critical role in web-based education as a considerable portion of the population is using online course delivery in their regular academic and information gathering activities. Concurrently, educational institutions also share a positive view on the emerging information and communication technologies and are adopting web-based education to reach a wider pool of learners. However, this new form of teaching and learning medium has created usability, accessibility and diversity issues in content delivery. Csikszentmihalyi introduced the concept of "being the flow" for optimal learning experience [11] and Bederson [12] adapted the concept to user interfaces and states that the user needs to concentrate on the task at hand and not be hindered by the interface. Interruption, literal or conceptual, affects the flow of learning and this gets in the way of users concentrating on their tasks. Therefore, in order to optimize the learning outcome, it is important to design teaching and learning tools with usability and accessibility in mind where computer systems work as a supporting and invisible tool. Diversity is an important parameter in this equation as well.

The study by Hadian [7] raised fundamental new issues in web based education: “In the quest for universal access and design for all-inclusive virtual classrooms, are we inadvertently creating barriers to other user groups that may also need to have access to the same virtual information? How do we target different users groups while maintaining optimal experience to each and every group? “ The study revealed that changeability is one design characteristic that is important in the definition of accessibility. In a virtual classroom, the author has to deal with different types of learners that require varying types and levels of accessibility and they have different style of learning. The fundamental question that the current research project attempts to address is: “How do we foster the development of learning and teaching tools that accommodate “diversity” in a virtual classroom while assuring “optimal" experience for all users?”

In theory, web pages can be updated frequently to provide customized versions on demand. In practice, however, rarely can authors provide multi-versioned web content to accommodate different levels of requirements by users. Also, the ability to maintain such sites is not feasible as HTML provides marginal support for the design and maintenance of multi-versioned sites. Teaching and Learning systems attempt to deal with customizing the content for user, but they provide “a version” of the page that is not the “best fit” of the content that meets the user’s need. Thus, in order to provide the user-centric optimal content, this project proposes the use of Intensional Logic originally proposed by Wadge and his co-workers [13,14]. Programming in a language based on intensional logic is called intensional programming [15]. This refers to programs that are sensitive to a global multidimensional runtime, which the program itself can modify. In the general case, an intensional program contains a number of versioned entities, each of which has a single valid instance under a particular global context. The set of (versioned entity, current context) pairs are termed an intension, whereas a specific, resolved version of the entity is termed an extension. As the program executes, Intensional entities are resolved to single versions against the global context.

5. BACKGROUND AND STATE OF THE ART

5.1. Learning Styles

Learning styles involve tailoring teaching methods to allow individuals to learn optimally. The idea of individualized "learning styles" originated in the seventies, and has gained popularity in recent years. It has been proposed that teachers should assess the learning styles of their students and adapt their classroom methods to best fit each student's learning style [16][17][18]. The different types of learning styles include content tailored for visual learners who need to see the instructor's body language and facial expressions to fully grasp the content of a lesson; auditory learners learn optimally by verbal lectures, discussions and listening; and hands-on students learn best by exploring the physical world around them. Methods for visual learners include ensuring that students can see words written down, using pictures when describing things, drawing time lines for events in history, writing assignments on the board, using overhead transparencies/handouts, and writing down instructions. Methods for auditory learners include repeating difficult words and concepts aloud, incorporating small-group discussion, organizing debates, listening to books on tape, writing oral reports, and encouraging oral interpretation. Methods for tactile learners include providing hands-on activities (experiments, etc.), assigning projects, having frequent breaks to allow movement, using visual aids and objects in the lesson, using role play, and having field trips. By using a variety of teaching methods from each of these categories, teachers are able to accommodate different learning styles.

There have been several models proposed to optimally deliver and/or transfer information to the learners such as the David Kolbe's model [19] based on experiential learning theory, Gregorc's model [20] is based on perceptions which in turn are the foundation of one's specific learning strengths, or learning styles. Individuals with different combinations learn in different ways—they have different strengths, different things make sense to them, different things are difficult for them. Sudbury Model democratic schools [21] assert that there are many ways to study and learn. This model asserts that there are many ways to learn without the intervention of teaching, to say, without the intervention of a teacher being imperative.

More recently, Chris J Jackson's hybrid model of learning argues sensation seeking provides a core biological drive of curiosity, learning and exploration. A high drive to explore leads to dysfunctional learning consequences unless cognitions such as goal orientation, conscientiousness, deep learning and emotional intelligence re-express it in more complex ways to achieve functional outcomes such as high work performance. This is a new model of learning and therefore remains in need of verification by independent research [22].

A non-peer-reviewed literature review by authors from the University of Newcastle upon Tyne identified 71 different theories of learning style [23]. This report, published in 2004, criticized most of the main instruments used to identify an individual's learning style. In this review, thirteen of the most influential models were selected for closer study. One of the most widely-known theories assessed was the learning styles model of Dunn and Dunn [24]. This model is widely used in schools in the United States. A recent report [25] provided evidence confirming the validity of Dunn and Dunn's model, concluding that matching students’ learning-style preferences with complementary instruction improved academic achievement and student attitudes toward learning. In their book, “Teaching Students through Their Individual Learning Styles: A Practical Approach”. [16], they give a background of how learners are affected by elements of the classroom and follow it with recommendations of how to accommodate students’ learning strengths. They analyze other research and make the claim that not only can students identify their preferred learning styles, but that students also score higher on tests, have better attitudes, and are more efficient if they are taught in ways to which they can more easily relate. Therefore, it is to the educator’s advantage to teach and test students in their preferred styles.

There are a plethora of models for learning styles. However, for the current research it is proposed to use the Dunn and Dunn's model which claims that visual learners have a preference for seeing (think in pictures; visual aids such as overhead slides, diagrams, handouts, etc.). Auditory learners best learn through listening (lectures, discussions, tapes, etc.). Tactile learners prefer to learn via experience—moving, touching, and doing (active exploration of the world; science projects; experiments, etc.). It is proposed that by designing web-based instructional content that accommodates each of these learning styles based on the individualized profile optimizes the web-based learning experience. This aspect addresses the diversity issues proposed in this research.

5.2. Teaching and Learning Content Management Systems

According to Owston [26], "access to learning" means making education more attainable by more people. When something is made accessible, it usually makes it more usable for everyone else. Accessibility also means that anyone using any kind of Web browsing technology must be able to visit any site and get a full and complete understanding of the information as well as have the full and complete ability to interact with the site. For example, when accessing poorly designed multimedia based Web content, some users cannot see graphics because of visual impairments; others cannot hear audio because of hearing impairments; and a large number of users have difficulty navigating sites that are poorly organized with unclear directions because they have learning disabilities.

A few research initiatives have been trying to address the accessibility issues on web-based information. The DO-IT (Disabilities, Opportunities, Internetworking and Technology), based at the University of Washington, is an organization that assists people with disabilities in successfully pursuing academics and careers, and offers workshops and educational outreach to promote the use of technology to maximize the independence, productivity and participation of people with disabilities [27].

EASI (Equal Access to Software and Information), based in Rochester New York, is a virtual organization that provides information and guidance in the area of access-to-information technologies by individuals with disabilities. Outreach programs include both on-site and on-line workshops, and use of web to research and disseminate information to colleges, universities, K-12 schools, libraries and into the workplace [28].

The SNOW (Special Needs Opportunity Windows) project is based at the University of Toronto. The goal of the SNOW project is to provide professional development and teaching resources to educators of students with special needs. This effort has focused on the development and moderation of on-line courses and discussion forums, dissemination of information and resource materials related to the education of students with special needs, and electronic delivery of curriculum resources in accessible formats [3][4][29].

In 1998, Bruce Landon looked at the pedagogical aspects of web based authoring tools [30], however, what was missing from his research was an assessment of the accessibility of these tools for learners with disabilities. The SNOW project extended this study with another in 2000, and concluded that these tools had somewhat improved the accessibility of web-based information, but further work was needed [Landon, 2000].

The NODE Learning Technologies Network, based in London, Ontario, is a not-for-profit electronic network, facilitating information and resource-sharing, collaboration and research in the field of learning technologies for post-secondary education and training. The NODE's Web site is a focal point for information and discussion forums on issues related to teaching, learning and technological development [31].

Since the late nineties, there has been a surge of concern regarding web accessibility by web developers [32, 33] resulting in guidelines and standardization [34]. Thus the American with Disabilities Act (ADA) of 1990 was amended to include the Section 508 of the Rehabilitation Act. In 2001, to make Section 508 enforceable, the Access Board published specific standards that spelled out what makes information (technology products, computer software, electronic office equipment and web-based services) accessible to people with disabilities, including those with vision, hearing and mobility impairment. The Canadian Government followed suit and reinforced section 508 in Canada [35] and introduced the Common Look and Feel (CLF) back in 2001.[36] Currently in Canada, Web sites launched after January 1, 2007 which are listed in Schedule I, I.1 and II of the Financial Administration Act [37] must conform to the CLF 2.0. standards [38]. Despite all the progress and improvement working toward accessible learning, challenges still remain in the areas of changeability.

To make web content accessible means: making the authoring and learning tools interoperable (functionality), adaptable (portability), operable and learnable (usability). The tools needs to have the ability to interact with specified operating systems (interoperability); and the tool needs to adapt to different environments without interference or actions other than those provided for this purpose (portability). For accessibility, the authoring tools also need to be operable to enhance the user's effort for operation and operation control; and learnable which bears on the user's effort for learning the application. The operability and learnability sub-characteristics determine how well the authoring tool deals with user interface issues, and the way the tool helps the author to navigate, locate, and learn about the availability of utilities that make the web material accessible to the user.

From the authoring perspective, the operability and learnability characteristics have the most influence in ensuring accessibility. Authoring tools allow an instructor to design and create instructional material for publishing on the web. Although most of these authoring tools may require HTML language knowledge, some programming or other technical expertise is needed to effectively use these tools: One such tool would be Dreamweaver.[39]

Learning and Content Management Systems (LCMS) are educational tools that manage, deliver and distribute learning and teaching content over the internet [40]. LCMSs are based on a variety of development platforms such as Java EE based architectures, PHP, and usually employ the use of a database back-end. LCMSs are multi-user environments where developers may create, store, reuse, manage, and deliver digital learning content (learning objects) from a central object repository. LCMSs require integrated authoring software in order to host the content. However, some of the LCMS also allowed authoring of web content. One such LCMS that has been popular during early 2000 and it was part of usability and accessibility test at UVIC was WebCT. [41] Today, WebCT or Blackboard Learning System, now owned by Blackboard, is an online proprietary virtual learning environment system. Instructors can add such tools as discussion boards, mail systems and live chat, along with content including documents and web pages. [42]

There are many authoring tools on the market, such Dokeos and Moodle. Dokeos was built corresponding to the traditional instructional design, and its structure is very close to traditional teaching (set of tools clearly marked as content creation tools), but clearly and simply extends this base by providing tools that encourage constructivism (forum, blogs, wikis, chat, file exchange, personal messaging, etc) [43].

Moodle is another free and open source e-learning software platform. Moodle is designed to help educators create online courses with opportunities for interaction. Developers can create additional functionality. Moodle has been evolving since the late nineties and more recent improvements include accessibility and user friendliness [44].

ATutor is an Open Source Web-based Learning Content Management System (LCMS). ATutor is used in various contexts, including online course management, continuing professional development for teachers, career development, and academic research. The software is cited as unique for its accessibility features, (useful to visually-impaired and disabled learners); and for its suitability for educational use according to software evaluation criteria established by The American Society for Training and Development (ASTD). ATutor is the first LCMS to comply completely with the accessibility specifications of W3C WCAG 1.0 at the AA+ level [45], allowing the access to all the included content of the system at all levels of user-privilege, including administrator accounts. ATutor can adapt to a wide variety of technologies including cell phones, personal data assistants (PDAs), and text-based Web browsers, to name a few[46].

Despite all the improvements and advances in computer science and related technologies, the currently available LCMSs do not have a true multi-versioning capability.. The authoring tools create several versions of the same content resulting in storage and maintenance issues. This is very costly and it does not truly address the issue of diversity with best fit to the user's learning style. It is noted that all the LCMS tools have made great strides in dealing with accessibility characteristics: operability, inter-operability, usability and learnability. However, changeability characteristic which deals with diversity of learning styles has not been addressed to date. In 2005, a study by Munoz [47] put forward the following question when comparing Blackboard to Moodle: "Did Blackboard/Moodle enhance instruction?" and the results were not encouraging. The current authoring tools have not been able to intensionalize the learning content to fit individual learning styles.

5.3. Intensional Programming Languages

The term "intensional" it is a relatively recent concept. Philosopher Carnap [48] introduced it in the thirties, based on Frege's [49] distinction between the "sense" and the "denotation" of an expression. Intensional logic is therefore the logic of expressions in which the intension of sub expressions (and not just their extensions) has to be taken into account.

The web must be intensional to optimally enable and enhance online interaction between authors and users. This is a radically different approach to the Web than that defined by Ted Nelson when he used the terms hypertext and hypermedia in 1965. Nelson presents a unipolar, extensional vision of hypermedia because he assumes the Web is a "widespread hypertext universe with ever-breaking one-way links" that has "no version management, no rights management, no parallel inter-comparison and no principled re-use of content materials" [50]. In other words, his versions are only collections of hyperlinked, independent extensions; they are not intensional renderings of a set of possible versions that can create a document. Alternatively, a database can be treated as a document, and databases do render versions of demand-driven documents, as any search engine demonstrates. However, these documents are rigidly structured and are not versionable.[51]

More to the point in the current context, programming in a language based on intensional logic is called intensional programming [52]. This refers to programs that are sensitive to a global multidimensional runtime, which the program itself can modify. With intensional languages, a versioned entity consists of a set of pairs of context-tagged extensional values (Ex, Cx), termed versions. The process of determining which version of an entity is used is termed a best-fit, and uses a partial order over the set of all contexts called refinement to compare the tag contexts of an entity with the global reference context Cr. The set of all tags in a versioned entity is termed a context domain [53].

The distinction between an intensional program and one that merely uses a context or pervasive data source as a polled reference for runtime configuration is that the semantics of an intensional program varies implicitly, often with language-level support for versioned control flow [54]. In the extreme case, anything with an identifier can exist in multiple versions, including all functions and data structures, as with the Perl-like language ISE [55].

In the general case, an intensional program contains a number of versioned entities, each of which has a single valid instance under a particular global context. The set of (versioned entity, current context) pair is termed an intension, whereas a specific, resolved version of the entity is termed an extension. As the program executes, intensional entities are resolved to single versions against the global context.

In 1979, Fieldman published a document outlining MAKE for maintaining software [56] In 1993, Wadge and Plaice created SLOTH as a new approach for controlling the multiple versions of a software. However, this original version of SLOTH was written in a monolithic manner and did not handle versions [52]. Subsequently, Brown coded LEMUR that allowed files to exist in multiple versions [50]. LEMUR was the result of intensionalizing SLOTH. From here, a subversion of LEMUR was created that allowed LEMUR to use not only the basic versions but sub-versions as well. Next, MARMOSET was created as an extension to LEMUR and this allowed different languages to be used using very simple configuration files and simpler than standard makefiles [57]. Hence, the idea of IML/ISE arrived in 1996. This envisioned parallel documents similar to Nelson’s Parallel Universe [58]. This was not a programming language and there was no provision for evaluating expressions. For instance, it could not do either constructs nor iterative constructs. Constructs refers to the basic elements, commands, and statements used in various programming languages. Subsequently, Wadge and his co-workers [59] proposed the Intensional HTML (IML/ISE) where the assembling work was done as a plug-in for the apache server. In 1999, this work was expanded and the Intensional Sequential Evaluator (ISE), a full-featured Perl-like scripting language that incorporated a (run-time) parametric versioning system, was created and written by Swoboda and Wadge [60]. Here, all the entities in the language such as variables, arrays and function could be versioned. At the same time, in 2000, Scraefel and Wadge [61] created the front end consisting of TROFF macros to encapsulate the Intensional Markup Language (IML). The macros were for stretching text and other constructs in an attempt to repair the deficiencies in HTML by making it practical to author web pages over a multidimension version space. The IML implementation used ISE, and it was a PERL like CGI language with run time parametrization.

In theory, digital web pages can be updated frequently and provided in customized versions on demand. In practice, however, this is rarely the case. The cost of delivering a new or customized version is negligible, but the maintenance is a real problem. HTML itself provides very little support for the production and maintenance of multi-versioned sites. The problem is that different versions of the same site cannot share pages except through links, but then they have to share all the pages accessible from the shared page. Consider a situation where a multilingual web site needs to be developed. For each language, the web site will have roughly the same structure and graphics. However, the text contained within each version of the web site will have to be different to reflect the chosen language. The conventional way to structure such a site is to create the English version, and subsequently translate the web content to a complete parallel version, one for each other language. In this scenario, each version would be a complete parallel site, with minimal reuse of the common structure of the web pages. Authors of multi-version sites are required to create copies by cloning, editing and maintaining the consistency of many separate but parallel pages. Wadge et al [58] proposed the Intensional HTML (IML/ISE) language which allowed the author to create source files for pages and parts of a page which, in general are generic, i.e. valid for a whole family of versions of the component in question. The structure is illustrated in Figure 3. The addition of a macro processing stage makes a difference in the usability of IML because it is possible to combine the parametric and modular approaches to diversity.

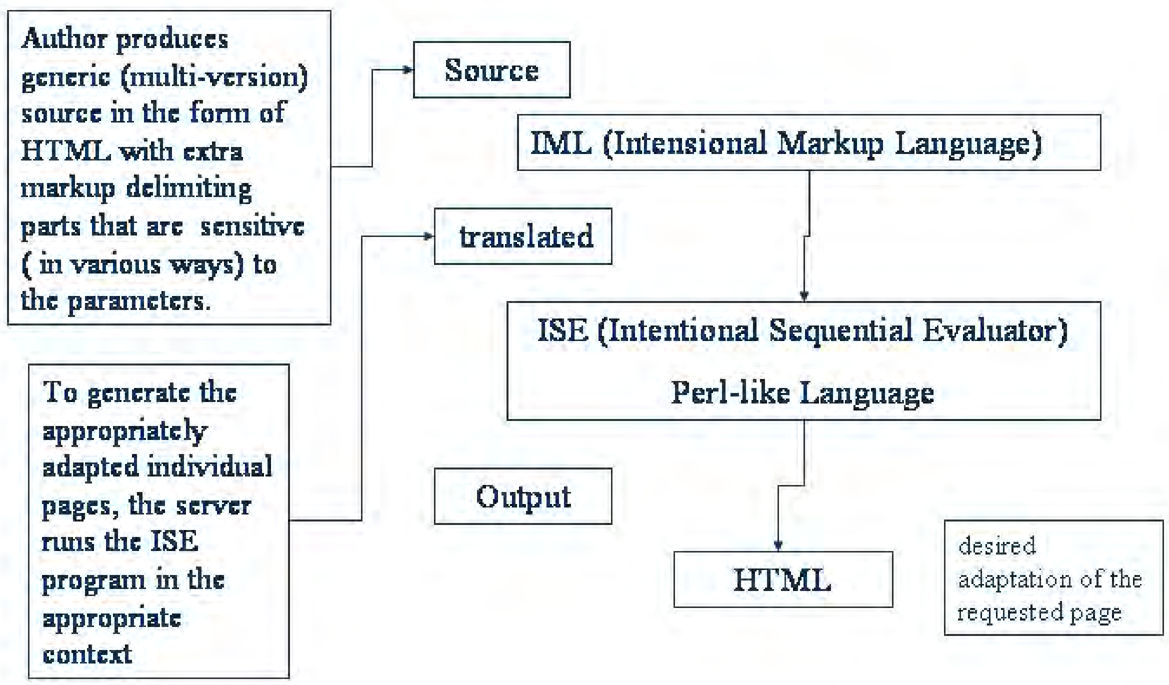

Author produces generic (multi-version) source in the form of HTML with extra markup delimiting parts that are sensitive (in various ways) to the parameters.
Source
IML (Intensional Markup Language)
translated
ISE (Intentional Sequential Evaluator)
Perl-like Language
To generate the appropriately adapted individual pages, the server runs the ISE program in the appropriate context
Output
HTML
desired adaptation of the requested page

Figure 3 – The structure of the Intensional Sequential Evaluator Program

IML/ISE was an extension to HTML and it allows versioning logic (intensional logic) to be applied within an HTML document. For example, each language represents a version in the dimension-space called language. IML/ISE allows the user to specify includes based on the version and select/switch statements (common to most programming languages) where the action to be performed in the document would vary depending on the current version. For a multilingual web site, one common set of documents could provide the overall structure, and IML/ISE's 'include' feature would include the appropriate language-specific section based on the current version. More recently, Swoboda et al [55] created the Intensional Sequential Evaluator (ISE) to overcome some of the limitations in IML/ISE such as the extra markup required for even simple effects such as droptext can be impractically complex when used in documents with a large number of droptext sections and the second disadvantage is that IML/ISE is not a programming language.

HTML itself provides very little support for the production and maintenance of multiversion sites. The problem is that different versions of the same site cannot share pages except through links; but then they have to share all the pages accessible from the shared page. Furthermore, pages with analogous layout cannot share the markup that describes the layout. In this study, it is proposed to address the issue of multiversioning using a language based on Intensional Logic called MMP (Markup Macro Processor). An authoring formalism is proposed in which one can specify whole families of related pages and sites with only a modest additional effort to that required to specify a single page or site. In a way, this study is an attempt to repair the deficiencies in HTML by making it practical to author web pages that vary over a multidimensional version space.

There is a major difference between this approach and other tools available currently. Based on a “best fit algorithm”, the user receives a unique version of the document that best fits the user’s request. In a virtual classroom, the author has to deal with different types of learners that require varying levels and types of accessibility. When the user can shrink or expand detail or shift the level of expertise within the current document window, that is, if it can make these adjustments without shifting contexts, it increases comfort, confidence and learning [61].

6. METHODOLOGY

6.1. Tool Components and Implementation

The implementation of the project is described in the Figure 4. This project consists of three inter-connected sub-projects that need to be dealt with in parallel and subsequently integrated. The core effort so far in the project is the multi-versioning issue, in order to be able to provide multiple versions of the same course content. This means that we need to move from the conventional linear environment to a multidimensional intensional programming environment. MMP is the software that was designed and implemented as an interpreter to enable the author to insert intensions into the HTML code. Presently, the development environment is only suitable for advanced programmers and average users have difficulty using it because HTML as not been designed with simplicity in mind. For this reason, Dreamweaver has been created to allow novice programmers to setup their own website. The second sub-project or issue that was tackled is the need to create a learners profile in order to describe the users learning abilities. The third component needs to deal with issues concerning storage and retrieval of the multi-versioned components.

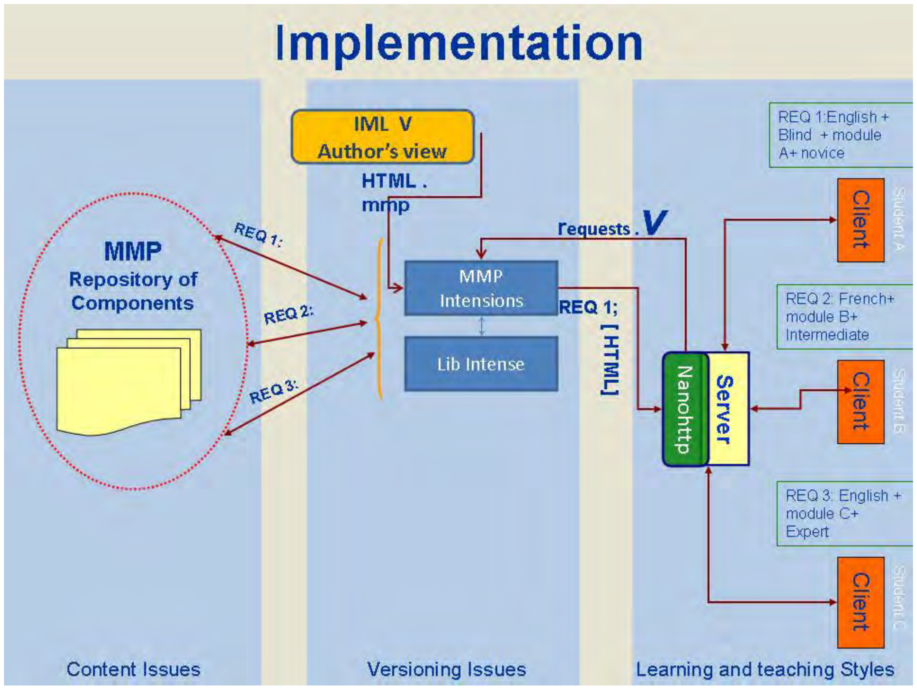


Figure 4 – The components for the project

Figure 5 describes the development stages for the MMP software. The first attempt to create a user interface to make it possible for the author to define and input the code for intensionalizing the content pages consists of combining troff macros and the requested versions resulted in the IML/ISE source file. This file was passed to the Apache server to create the HTML file. The first attempt resulted in a lack of functionality. This version of IML/ISE did not make provisions for intensionalizing functions or providing any of the programming features such as structs and looping. It also did not allow for extensions. In the second version, the algorithm for locating the appropriate learning object was written in Perl because of it fast text capability. GROFF macros were used to create the HTML code plus the versions, the interface that users would use to program their pages. The lack of functionality and extensibility of the version II led to the development of Intensional Markup Language (IML). In version III, the author creates a source file that includes groff macros and html code. The Intensional Sequential Evaluator (ISE) enabled the creation of structs and variables allowing intensionalizing the content. Once the versioning requirements are defined by the user, the ISE interpreter uses a "best fit algorithm" to package the content in an optimal fashion. This implementation improved functionality and some of the usability but still lacked extensibility. Subsequently, in version III Perl was used as a scripting language that was able to scan trough text very fast. Also the libintense (library of predefined macros) was created to enable an object modeling. The request and versions came from the user and the Perl interpreter performs the best fit algorithm to match the optimal component and looked up for the libintense for the macro definition. This version improved the extensibility and also made usability easier.

Finally, with the focus on simplicity for authoring and functionality of the tool, the Markup Macro Processor (MMP) was created to replace the two steps needed to create the .n and .m files. The MMP file interprets the macro definitions. This makes the steps simpler and possible to increase functionality of the tool as a programming language. However, the load on the server was an issue. Apache as web server opens a session for every request it receives, so the bottleneck of data transmission becomes an issue; therefore, it was replaced by nanohttpd, a light weight web server nanohttpd is written in Java, so once we start an instance of the server, it keeps running. It listens to any incoming traffic and sends requests through the MMP and code gets processed at runtime and gets returned to the browser (user) as an HTML document. The effect of nanohttp in high traffic area is still unknown. But in moderate traffic it has performed very well.

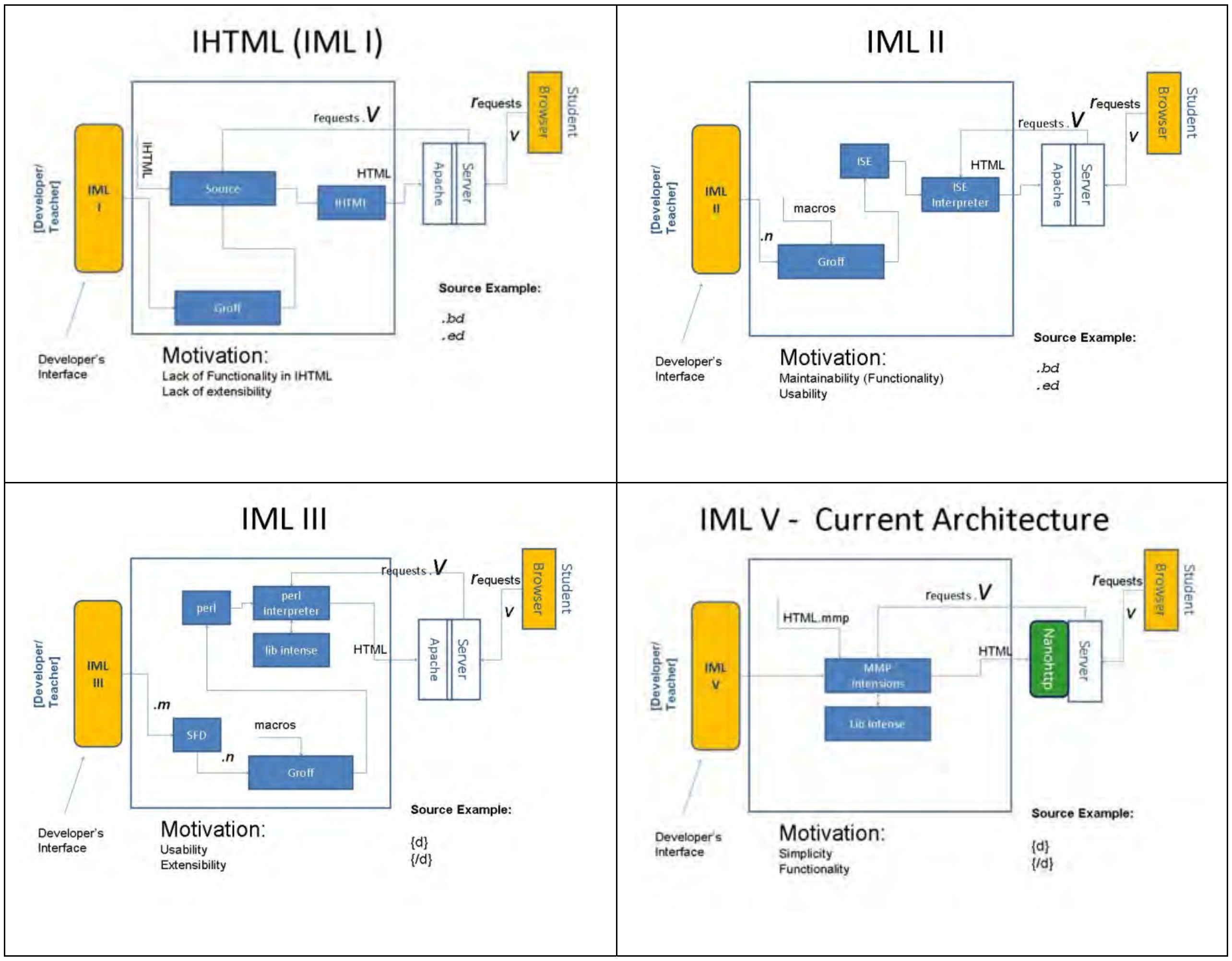


Figure 5 – The development stages for the creation of the HTML files using Intensional Markup Language (IML)

The design and implementation of the CMS(Content Management System) was based on Software Engineering Design principles to create a site that is modular and extendable. The CMS site was designed with four goals in mind:

i. A portal for our demo and testing site
ii. The home of the "online Learning Community Project for users with Cognitive disabilities, in collaboration with UVIC-CANAssist.

iii. It provides a secured environment for collaborators of this project and archive of all available report or presentations.

iv. To design the DEAL site with software design principle and intensionalize it, so we can use the site as a prototype for our testing

The implementation of the new framework consisted of three parallel processes: the development of a new interpreter language, profiling of the learning abilities, the development of the learning module such as a lesson, and the collection of the learning contents for optimal delivery based on the ability. A new language interpreter, referred to as the Markup Macro Processor (MMP) was developed and implemented. This is the core of the effort in the proposed research project as it is the missing link that allows intensions to be inserted into the HTML code. These intentions in turn allow the creation of a multi-dimensional content space. The language is still in the development phase.

The core component of the project is the multi-versioning issue, in order to be able to provide multiple versions of the same course content. This means that we moved from the conventional linear environment to a multidimensional intensional programming environment. The Markup Macro Processor (MMP) is the software that was designed and implemented as an interpreter to enable the author to insert intensions into the HTML code. Presently, the development environment is only suitable for advanced programmers and average users have difficulty using it because HTML as not been designed with simplicity in mind.

This research will enable the teacher to develop curricula that reaches all students regardless of their physical needs or learning abilities. An added-value of the findings and conclusions of the proposed study with be a clearer understanding of the needs for more equitable access to workplace-related training for older workers, persons with disabilities, workers in remote and rural areas, immigrants and those with low literacy skills, and by groups such as community-based organizations.

6.2. Implementation of a Test Case

The current example describes the implementation of the multiversioning idea proposed in this research. In order to better understand the implementation of the tool, let us discuss Figure [6]

Let us assume that we have three clients connected to the server. The nanohttp is a light java based webserver selected over the apache server because there is only copy of the server running at any one time while the apache server opens a new thread every time a request arrives. With the multiversioning implementation, if a new thread is opened for every request, it would create a backlog of open threads sitting on the server. Based on the user's request, the server sends the request version to the MMP interpreter which in turn opens the macros and then inserts the requested versions into the HTML code and in turn returns to the web server for rendering to the user.

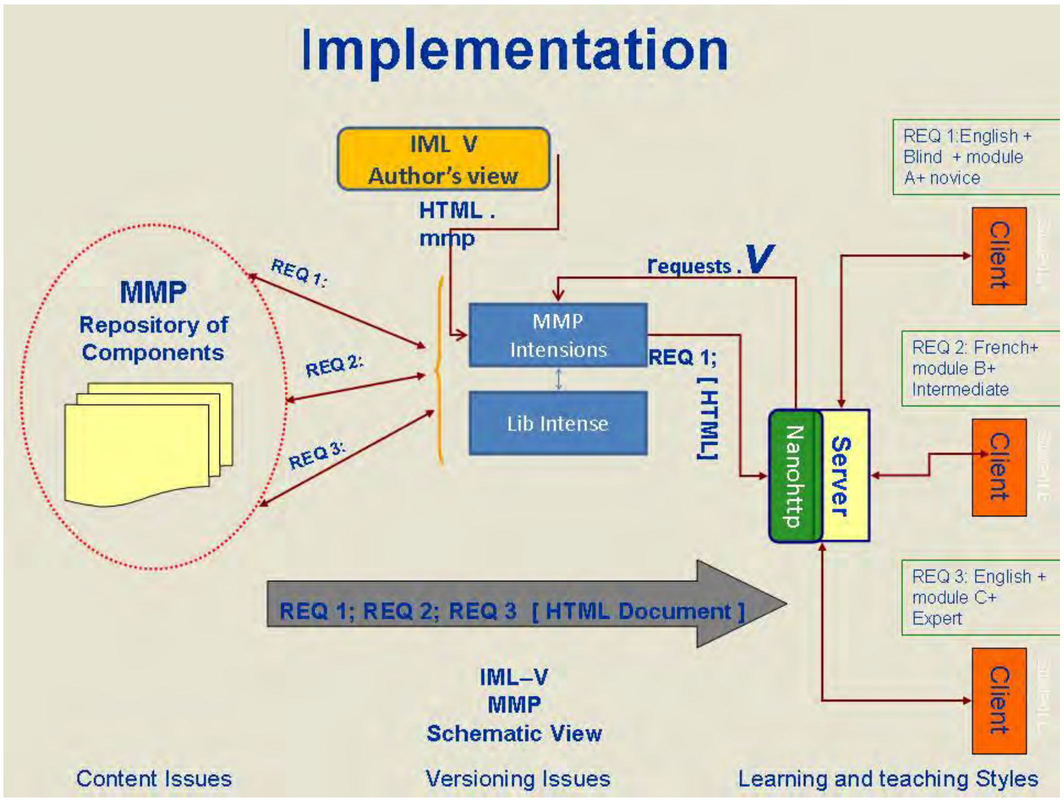


Figure 6 - The schematic view of the tool for rendering multiversioned content to the user upon requests sent through the server

Consider a web site that has more than one dimension: a multi-lingual reference manual for an automobile, with low-bandwidth (for slow internet connections) and high-bandwidth (for fast internet connections) versions. There would be many dimensions here: language, automobile model number, model-year, specific options, and low or high bandwidth. Creating this web site using the conventional approach would result in an exponential blowup of the number of different versions that would need to be created and maintained.

Creating the site using ISE would at least be possible since the problem itself can be simplified by partitioning the versions into different dimensions, and then tackling versions within each dimension. ISE uses a modified version of the Apache Web Server. The HTML pages that are sent to the client's browser are in the form of standard HTML. The IML/ISE markup is interpreted by the Web Server (CGI scripts or Apache), and then HTML is dynamically generated based on the version space. Besides multiversioned web pages, IML/ISE can be used to create new forms of hypertext.

It is not necessary to provide files for each version that the user might request. The server uses a best fit algorithm to choose the version that most closely approximates the requested one. If the requested version is available, it will be used. Otherwise, if there is a unique closest match, it is used. I had to modify my idea into a tree structure, where I created a menu to narrow down the Selection of disability and types of products.

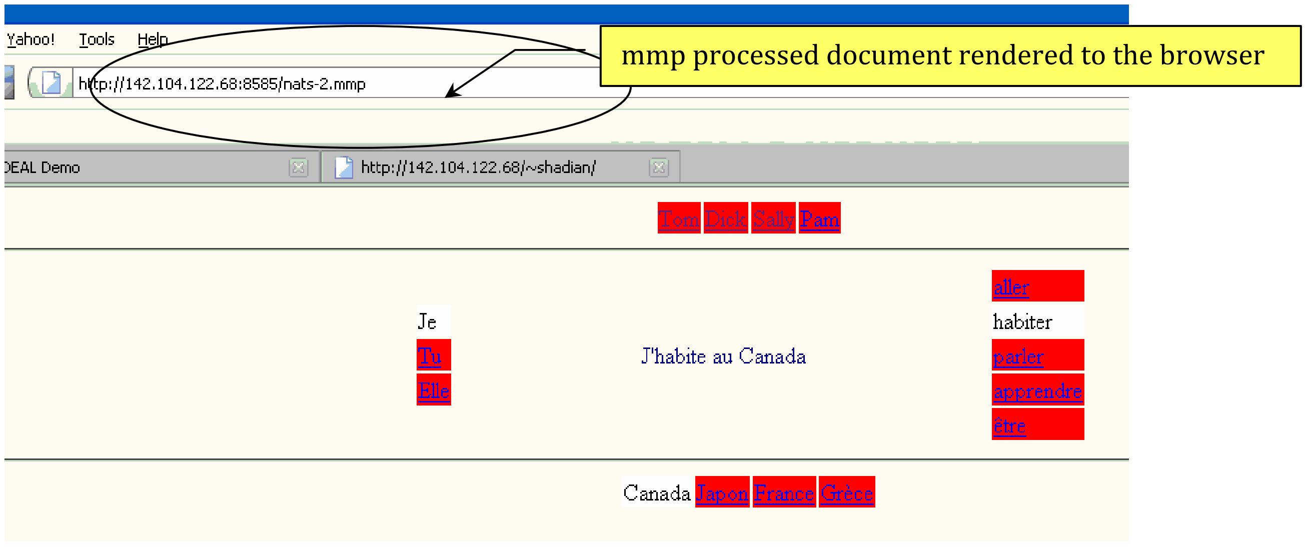
Once the user gets the list of results, connect them to their individual files. These files contain the information which resembles the fields of the records. The structure of digital documents was used to format the files. Indexes for navigation and appropriate spacing were used to make sure that the information is readable.

Besides multiversioned web pages, IML/ISE can be used to create new forms of hypertext. For example, pop text used in the IML/ISE user manual link below was created with the aid of IML/ISE. I planned to use a search engine that would access two data bases one holding all the Information required for disabled users and the other of all the available assistive technology. Idea was to create a dynamic web page that would query these two data bases and would narrow down the options for the user. They could query that based on their disability and what they require which products are available and some of their features.

To organize this information in the context of this course became a challenge as the size of the data base is finite numbers, the information has to be organized in a file rather than a database, and the 2-D vision of arranging this information and accessing them has to be fit a linear model.

It is not necessary to provide files for each version that the user might request. The server uses a best fit algorithm to choose the version that most closely approximates the requested one. If the requested version is available, it will be used. Otherwise, if there is a unique closest match, it is used.

Let us look at the example in Figure 7. The upper part includes a group of people (male and female). On the left hand side it includes the nouns (singular and plural). On the right hand side we have the verbs (aller, habiter, parler, apprendre, etre). At the bottom, we have the countries. Based on the user's request, the central frame represents a customized output. This frame displays the user's preference based on the input selected. For example, if the user selects je, habiter and Canada, the central frame automatically displays j'habite au Canada. If the user selects , Nous, Sally and Pam, aller and Grece, the display automatically shows Nous allons en Grece. The server is delivering a best fit customized output based on the request selected by the user. If the tool provides ways to provide multiple versions of the content, then it eliminates the need to create multiple version of the same site. The source code for this simple example is included here. Authors create multiple labeled versions of the IML/ISE source for a given page. In turn, requests from clients specify both a page and a version. Next, the server software selects the appropriate source page and uses the source page to generate the requested actual html page. If server-side software cannot find a source page with the exact version, it uses the page whose label most closely approximates the requested version. In turn, it treats the refinement ordering as a (reverse) inheritance ordering which means that different versions can share source, authors can write generic, multi-version code.



http://142.104.122.68:8585/nats-2.mmp

http://142.104.122.68/~shadian/

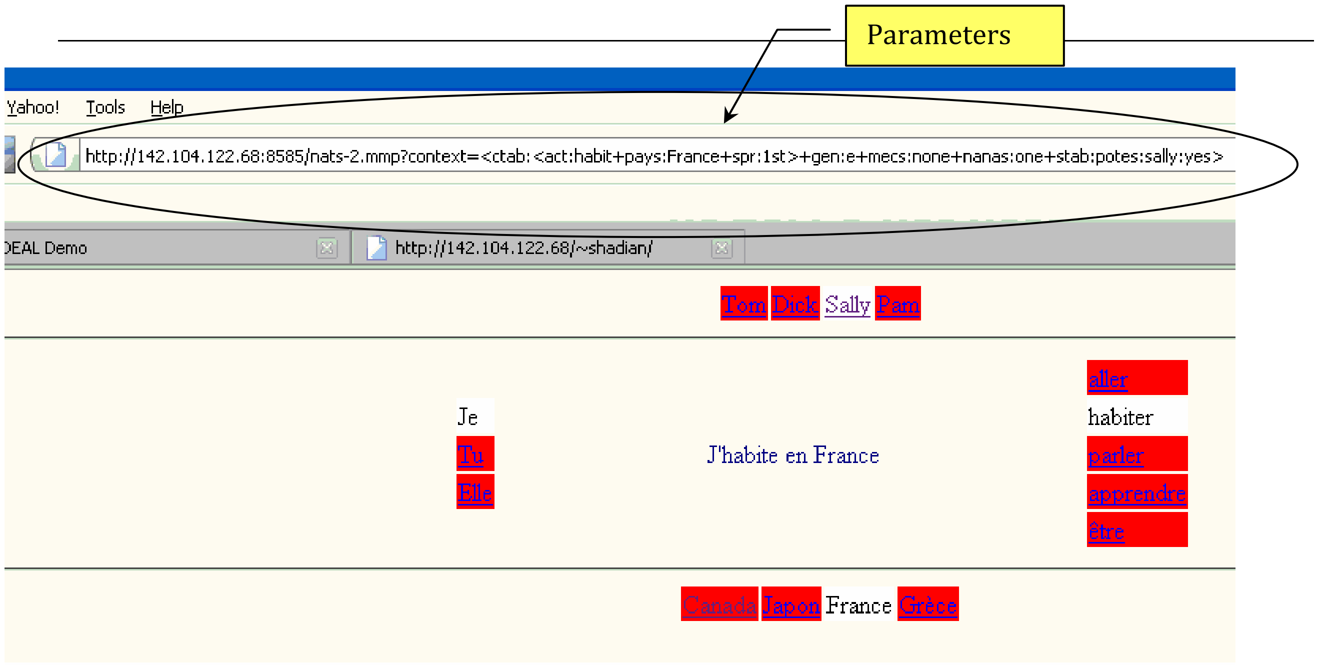
Tom Dick Sally Pam

Je
Tu
Elle

J'habite au Canada

aller
habiter
parler
apprendre
être

Canada Japon France Grèce



http://142.104.122.68:8585/nats-2.mmp?context=<ctab:<act:habit+pays:France+spr:1st>+gen:e+mecs:none+nanas:one+stab:potes:sally:yes>

http://142.104.122.68/~shadian/

Tom Dick Sally Pam

Je
Tu
Elle

J'habite en France

aller
habiter
parler
apprendre
être

Canada Japon France Grèce

Figure 7 - Example of Implementation

6.3 Source MMP File

```
{include html.mmp/}                    // link to another macro

{" : web}<html>                        // embedded macro
<title>{1/}</title>
<body>
{.../}
</body>
</html>{/"}

{" : emph}<i>{bold}{.../}{/bold}</i>{/"}
{" : bold}<b>{.../}</b>{/"}

{web "Les nationalit&eacute;s"}     //macro call

{@: tdir}{/@:}
{@: tab}{/@:}

{" : initdim}{v}{" , ""}{[] {1/}:{2/}/}{/"}{" , {1/}:~}{/"}{/v}{/"}
{: so}{/:}
{: def}{/:}
{" : ctc}{cell{@ tdir/}}{button {1/} {2/}/}{/cell{@ tdir/}}{/"}
{" : stc}{cell{@ tdir/}}{sbutton {1/} {2/}/}{/cell{@ tdir/}}{/"}
{" : sbutton}<td bgcolor=white>{2/}</td>{/"}

{" : sbutton}
 {v}
  {" , ""}<td bgcolor=red>{alink}{2/}{/alink stab:{@ tab/}:{1/}:yes}
</td>{/"}
  {" , stab:{@ tab/}:{1/}:yes}<td bgcolor=white>{alink}{2/}{/alink stab:{@
tab/}:--+stab:{@ tab/}:{1/}:yes}</td>{/"}
 {/v}
{/"}


{" : button}
 {v}
  {" , ""}<td bgcolor=red>{alink}{2/}{/alink ctab:{@ tab/}:{1/}} </td>{/"}
  {" , ctab:{@ tab/}:{1/}}<td bgcolor=white>{2/}</td>{/"}
 {/v}
{/"}

{" : cell}<tr>{_/}</tr>{/"}
{" : cellh}{_/}{/"}
{" : ctab}{@= tab}{1/}{/@=}{@= tdir}{2/}{/@=}{",/}<table>{_/}</table>{/"}
{" : stab}{@= tab}{1/}{/@=}{@= tdir}{2/}{/@=}{",/}<table>{_/}</table>{/"}


{" : def}{@: {*/}}{_/}{/@:}{/"}
{: dimexp}{/:}
{: incp}{/:}
{" : val}{@ {*/}/}{/"}
{: sval}{/:}
```

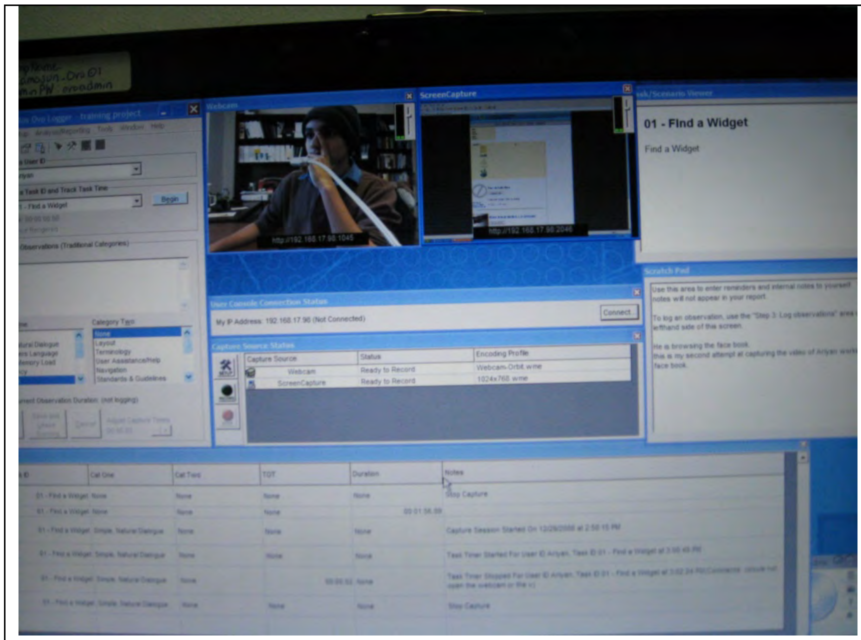
```
 {initdim ctab:pays Canada/}
 {initdim ctab:act habit/}
 {initdim ctab:spr 1st/}

 {[] mecs:-+nanas:-+num:-+gen:-/}

 {cvm stab:potes:<dick:yes+tom:yes> mecs two/}
 {cvm stab:potes:<dick:yes> mecs one/}
 {cvm stab:potes:<tom:yes> mecs one/}
 {initdim mecs none/}

 {cvm stab:potes:<pam:yes+sally:yes> nanas two/}
 {cvm stab:potes:<sally:yes> nanas one/}
 {cvm stab:potes:<pam:yes> nanas one/}
 {initdim nanas none/}

 {cvm mecs:two num s/}
 {cvm nanas:two num s/}
 {cvm mecs:one+nanas:one num s/}

 {cvm mecs:none gen e/}

{def fpp}Je{/def}
{def fpp num:s}Nous{/def}

{def spp}Tu{/def}
{def spp num:s}Vous{/def}

{" def tpp}Il{# num/}{/"}
{" def tpp gen:e}Elle{# num/}{/"}

{incp pays:Canada act:parl/}
{incp pays:Canada act:appren/}

{" def suj ctab:spr:1st}{val fpp/} {/"}
{" def suj ctab:spr:2nd}{val spp/} {/"}
{" def suj ctab:spr:3rd}{val tpp/} {/"}
{" def suj ctab:<spr:1st+act:appren>}J'{/"}
{" def suj ctab:<spr:1st+act:habit>}J'{/"}
{" def suj ctab:<spr:1st+act:appren>+num:s}Nous {/"}
{" def suj ctab:<spr:1st+act:habit>+num:s}Nous {/"}


{" def vph}{# ctab:act/}{val term/}{/"}
{" def vph ctab:<act:etre+spr:1st>}suis{/"}
{" def vph ctab:<act:etre+spr:2nd>}es{/"}
{" def vph ctab:<act:etre+spr:3rd>}est{/"}
{" def vph ctab:<act:etre+spr:1st>+num:s}sommes{/"}
{" def vph ctab:<act:etre+spr:2nd>+num:s}&ecirc;tes{/"}
{" def vph ctab:<act:etre+spr:3rd>+num:s}sont{/"}

{" def vph ctab:<act:all+spr:1st>}vais{/"}
{" def vph ctab:<act:all+spr:2nd>}vas{/"}
{" def vph ctab:<act:all+spr:3rd>}va{/"}
{" def vph ctab:<act:all+spr:1st>+num:s}allons{/"}
{" def vph ctab:<act:all+spr:2nd>+num:s}allez{/"}
{" def vph ctab:<act:all+spr:3rd>+num:s}vont{/"}

{" def vph ctab:<act:appren+spr:1st>}apprend{/"}
{" def vph ctab:<act:appren+spr:2nd>}apprends{/"}
{" def vph ctab:<act:appren+spr:3rd>}apprend{/"}
{" def vph ctab:<act:appren+spr:1st>+num:s}apprenons{/"}
{" def vph ctab:<act:appren+spr:2nd>+num:s}apprenez{/"}
{" def vph ctab:<act:appren+spr:3rd>+num:s}apprennent{/"}

{" def cmp ctab:act:etre}{val adje/}{/"}
{" def cmp ctab:act:parl}{val lang/}{/"}
{" def cmp ctab:act:parl+ctab:pays:Canada}fran&ccedil;ais et anglais{/"}
{" def cmp ctab:act:appren+ctab:pays:Canada}le fran&ccedil;ais et
l'anglais{/"}
{" def cmp ctab:act:appren}le {val lang/}{/"}
{" def cmp ctab:act:all}{val dest/}{/"}
{" def cmp ctab:act:habit}{val dest/}{/"}

{" def dest}en {# ctab:pays/}{/"}
{" def dest ctab:pays:Grece}en Gr&egrave;ce{/"}
{" def dest ctab:pays:Canada}au {# ctab:pays/}{/"}
{" def dest ctab:pays:Japon}au {# ctab:pays/}{/"}

{" def adje ctab:pays:Canada}canad{val ien/}{/"}
{" def adje ctab:pays:Japon}japon{val ais/}{/"}
{" def adje ctab:pays:Grece}grec{# num/}{/"}
{" def adje ctab:pays:Grece+gen:e}grecque{# num/}{/"}
{" def adje ctab:pays:France}fran&ccedil;{val ais/}{/"}

{def ais}ais{/def}
{" def ais gen:e}aise{# num/}{/"}

{" def ien}ien{# num/}{/"}
{" def ien gen:e}ienne{# num/}{/"}

{" def lang}{! num:-+gen:-}{val adje/}{/!}{/"}


{def term}e{/def}
{def term num:s}ent{/def}
{def term ctab:spr:2nd}es{/def}
{def term num:s+ctab:spr:1st}ons{/def}
{def term num:s+ctab:spr:2nd}ez{/def}

<body bgcolor="EAEAEA">
<center>
{stab potes h}
 {stc tom Tom/}
 {stc dick Dick/}
 {stc sally Sally/}
 {stc pam Pam/}
{/stab}
<hr>
<table border=0>
<tr>
<td>
```

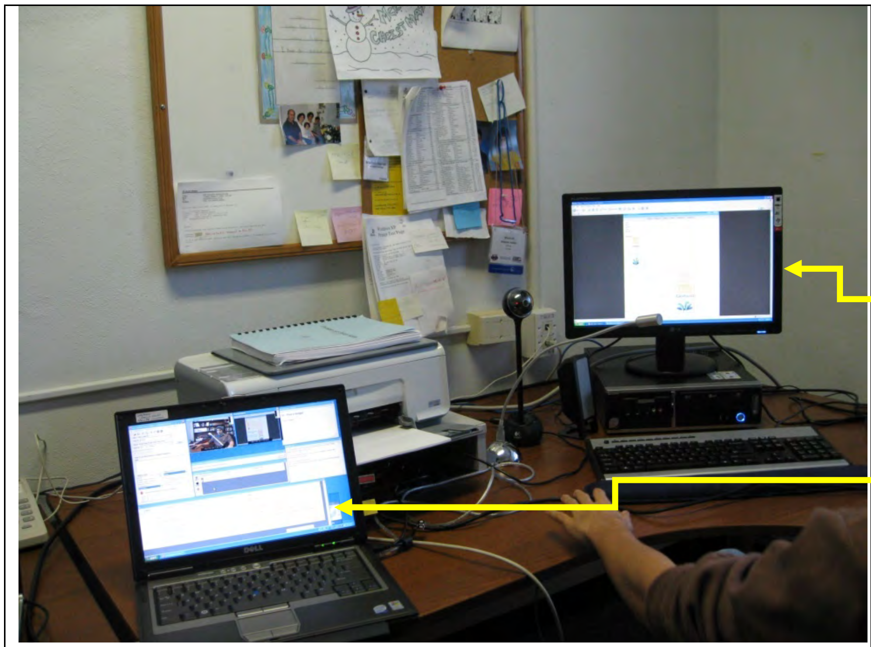
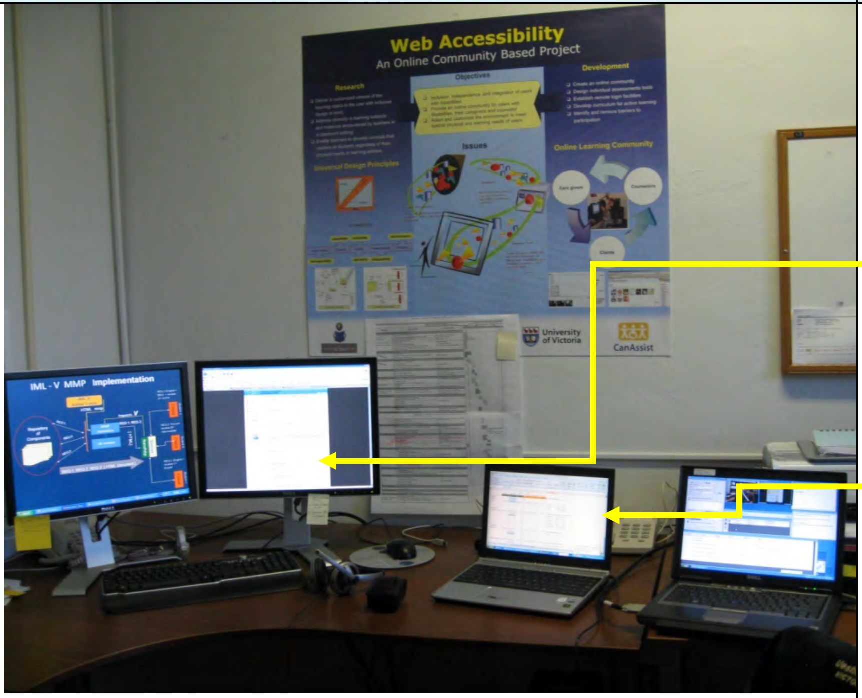
```
<table border=0>
<tr>
<td>

 {ctab spr}

  {ctc 1st {val fpp/}/}
  {ctc 2nd {val spp/}/}
  {ctc 3rd {val tpp/}/}

{/ctab}


 </tr></table>

   </td>
   <td align=center width=360>

   <font color="000099">

     {val suj/}{val vph/} {val cmp/}

   </font></td>
   <td>

 {ctab act}

  {ctc all aller/}
  {ctc habit habiter/}
  {ctc parl parler/}
  {ctc appren apprendre/}
  {ctc etre &ecirc;tre/}

 {/ctab}

  </td>
  </tr>
 </table>
 <hr>

{ctab pays h}

 {ctc Canada Canada/}
 {ctc Japon Japon/}
 {ctc France France/}
 {ctc Grece Gr&egrave;ce/}

{/ctab}

</center>
</body>

{/web}
```

html.mmp Linked Macro Source Code

```
{" : link}<a href={1/}?context={en}{! {-1/}}{#/}{/!}{/en}>{.../}</a>{/"}

{" : alink}{link "{% URI/}"}{_/}{/link {-1/}}{/"}

{" : en}{r "<" "%3C" ">" "%3E"}{.../}{/r}{/"}

{" : h}<a href={1/}.mmp>{<</}{*/}</a>{/"}

{" : ph}<a href={1/}.mmp?{<</}{ap{a#/} {*/}/}>{.../}</a>{/"}

{" : ap0}{/"}
{" : ap1}1={1/}{/"}
{" : ap2}1={1/}&2={2/}{/"}
{" : ap3}1={1/}&2={2/}&3={ap3}{/"}

{" : r$}{r {"}\$1 "{1/}" \$2 "{2/}" \$3 "{3/}" \$_ "{_/}"{/"}}{_/}{/r}{/"}

{" : de}{: {1/}}{r$}{_/}{/r$}{/:}{/"}

{" de cvm}
 {v}
  {" , $1}{[] $2:$3/}{/"}
  {" , $1+$2:~}{/"}
  {" , ""}{/"}
 {/v}
{/"}

{" : html#}{r \< \&lt; \> \&gt;}{#/}{/r}{/"}
```

7. THE CC USABILITY AND ACCESSIBILITY LABORATORY (CCUAL)

 01 - Find a Widget Find a Widget ScreenCapture Webcam	Our **portable usability lab** consist of the following components: 1) Ovo Logger on a notebook computer. This is the core of the portable lab and enables the following functions: • Logging the usability test • Recording camera output • Recording the user's computer screen • Making video highlights • Writing tester's report • Exporting tester's report in HTML format. Screen recording hardware, for recording the user's computer screen in native XGA (up to 1600x1200.) All necessary hardware and accessories including webcam (for recording the user's face), VGA splitter, LAN hub, USB microphone, power strip, all interconnecting cables.

	**OVO Studios Portable Lab** 1) Consists on a laptop for recording of audio/video of one other participant workstation/laptop 1 dual core PC station 1 - 20 inch monitors Dell Laptop with OVO Logger 6.0 Uses a portable webcam and microphone for recording
 Web Accessibility An Online Community Based Project IML - V MMP Implementation	**Development Station** The following equipment has been obtained with the future goal of converting them to a fixed lab. 1 Dell Precision workstation 490 By installing a copy of the OVO Logger X, It can monitor 4 -8 monitors at the same time. 2 - 20 inch monitors **Other equipment** 1 Toshiba tablet 1 **Server** – For dissemination of Information and hosting the prototype tool ● Two Dual Core AMS processors ● 2X2GB Dual Ranked DIMM memory chip ● Linux Operating System

8. CONCLUSIONS

A LCMS (Learning Content Management System) called *Diversity in an Environment for Accessible Learning* (DEAL) was designed and implemented for the dissemination of information and also to extend the LCMS as a prototype tool to be intensionalized based on the MMP interpreter.

The challenge of dealing with diversity issue of users and the ability of a teaching and learning tool to deal with the issue of changeability is a daunting task. Software engineers and designers have developed tools like *ajax* and the *javascript* language to make the web site more dynamic for the user, but to fundamentally change the web interface philosophy to make it suitable to a diverse universe of users according to their learning profiles has not been accomplished to date. The multi-versioning capability in web software design is still in its infancy. This project, by adding the concept of intensional logic, within the framework of the proposed accessibility definition and the current technology, and by taking into account the accepted models of learning styles will advance the state of the art by providing an authoring tool that enables the teachers to incorporate different learning styles to address diversity in their virtual classroom. .

The collaborations with established research institutions such as UVIc have been another major impact from the college's perspective. The teaching and learning tool, when completed, will prove to be beneficial to the community and teaching and learning as a whole. With the focus of this project on teaching and learning, we are hoping that it will let policy making bodies to notice the importance of this project in advancing the Usability and Accessibility of Web-based educational content.

The development site and all of the reports and information are posted online. The information can be accessed using the following login parameters: The DEAL website can be found at the following URL: http://142.104.122.68/framework/ and the original Online Learning project information is posted on the following website: http://142.104.122.68/OnlineLearning/index.php
.

REFERENCES

[1] Learning Technology Standards Committee, 2002, Draft Standard for Learning Object Metadata. IEEE Standard 1484.12.1, http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf, retrieved on 2009-06-29.

[2] Wiley, D. A. (2000). "Connecting Learning Objects to Instructional Design Theory: A Definition, a Metaphor, and a Taxonomy". In D. A. Wiley (Ed.), The Instructional Use of Learning Objects: OnlineVersion. http://reusability.org/read/chapters/wiley.doc.

[3] ATRC Staff, 1998, "Alternative Keyboard and other Enhancements". Retrieved from the Web", June 29, 2009 http://atrc.utoronto.ca/index.php?option=com_content&task=view&id=32&Itemid=9

[4] L. Harrison, 2000 ,"Inclusion in an Electronic Classroom - 2000: the Role of the Courseware Authoring Tool Developer", Adaptive Technology Resource Centre, University of Toronto, http://www.atutor.ca/research/access_study/inclusion.html

[5] Nielsen N., 2001,"Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities", Nielsen Norman Group, http://www.nngroup.com/reports/accessibility/

[6] Storey M-A. D, Phillips B., M. Maczewski B. and M. Wang, 2002, "Evaluating the Usability of Web-based Learning Tools" , Educational Technology and Society, Vol 5, No. 3.

[7] S. Hadian, 2004,"Accessibility Issues in Web-Based Education: a Case Study for the Visually Impaired", M.Sc. Thesis, University of Victoria, Department of Computer Science, Victoria, B.C., Canada.

[8] Hricko M., 2003, "Design and Implementation of Web-Enabled Teaching Tools", Information Science Publishing

[9] A. Foley A. , 2003, "Distance, Disability and the Commodification of Education: Web Accessibility and the Construction of Knowledge" , Journal of Current Issues in Comparative Education, Vol. 6, No. 1

[10] ESSI-SCOPE, 2000, "Information Software Product Quality", http://www.cse.dcu.ie/essiscope/sm2/9126ref.html

[11] Csikszentmihalyi M., 1991, Flow: the Psychology of Optimal Experience , Harper Collins.

[12] B. Bederson B., October 2003, "Interfaces for Staying in the Flow", Technical Report HCIL-2003-37, University of Maryland,

[13] Schraefel, M.C., 1997, "ConTexts, Intensional Document Creation, Delivery and Retrieval", Communications, Computers and Signal Processing,;10 Years - IEEE Pacific Rim Conference, Volume 1, Issue , 20-22 Aug 1997 Page(s):417 – 419.

[14] Wadge W., 2003, “Intensional Markup Language”, Lecture Notes in Computer Science; also Proceedings of the Distributed Communities on the Web: Third International Workshop, 2000, Quebec City, Canada, Vol. 1830/2000

[15] Faustini A.A. and Wadge W.W., 1987, “Intensional programming, The Role of Languages in ProblemSolving”, J.C. Boudreaux, B.W. Hamil and R. Jenigan, eds., 2, Elsevier North-Holland.

[16] Dunn, R, & Dunn, K ,1978, “Teaching students through their individual learning styles: A practical approach”. Reston, VA: Reston Publishing Company.

[17] LdPride, n.d.. What are learning styles? Retrieved June 26, 2009; http://www.ldpride.net/learningstyles.MI.htm#Learning%20Styles%20Explained

[18] Sprenger, M., 2003, “Differentiation through learning styles and memory”. Thousand Oaks, CA: Corwin Press

[19] David Kolb, 1984, “Experiential learning: Experience as the source of learning and development”. Englewood Cliffs, NJ: Prentice-Hall

[20] Mills, D. W., 2002,”Applying what we know: Student learning styles”. Retrieved June 26, 2009, from: http://www.csrnet.org/csrnet/articles/student-learning-styles.html

[21] Greenberg, D., 1992, Education in America, A View from Sudbury Valley, "Special Education" -- A Noble Cause Run Amok.

[22] Jackson, C. J.,2008, “Measurement issues concerning a personality model spanning temperament, character and experience”. In Boyle, G., Matthews, G. & Saklofske, D. Handbook of Personality and Testing, Sage Publishers. (pp. 73-93)

[23] Coffield, F., Moseley, D., Hall, E., Ecclestone, K, 2004, “Learning styles and pedagogy in post-16 learning: A systematic and critical review”. London: Learning and Skills Research Centre.

[24] Dunn, R., Dunn, K., & Price, G. E., 1984, “Learning style inventory”. Lawrence, KS, USA: Price Systems

[25] Lovelace, MK , 2005, “ Meta-Analysis of Experimental Research Based on the Dunn and Dunn Model”. Journal Of Educational Research, 98: 176-183

[26] R. D. Owston R.D., 1997, “The World Wide Web: A Technology to Enhance Teaching and Learning? ” Educational Researcher, Vol. 26, No.2, P.27-33.

[27] S. Burgstahler S., 2000, Disabilities, Opportunities, Internetworking and Technology, http://weber.u.washington.edu/\~doit

[28] EASI, Equal Access to Software and Information , 2000, Rochester New York, http://www.isc.rit/edu/\~easi/

[29] SNOW - Special Needs Opportunity Windows, 2000, http://snow.utoronto.ca/

[30] Landon B. and Robson R., 1999, “Technical Issues in Systems for WWW-Based Course Support”, International Journal of Educational Telecommunications, Vol.5, No.4, P. 437 -453.

[31] The Node learning Technologies Network, 2001, http://Thenode.org/

[32] World Wide Web Consortium, 2000, http://www.w3.org

[33] Web Accessibility Initiative, 2000, http://www.w3.org/WAI/

[34] Section 508 United States, 2000, http://www.section508.gov/index.cfm?FuseAction=content&ID=12; retrieved June 29, 2009.

[35] Section 508 Canada, 2001, http://www.canadainternational.gc.ca/sell2usgov-vendreaugouvusa/opportunities-opportunites/section_508.aspx?lang=eng; retrieved June 29, 2009.

[36] CLF 1.0: Common Look and Feel for the Internet, 2001, Government of Canada, http://www.tbs-sct.gc.ca/clf-nsi/inter/inter-01-01-eng.asp

[37] Financial Administration Act , 2006, Government of Canada, http://laws.justice.gc.ca/en/F-11/index.html

[38] CLF 2.0: Common Look and Feel, 2006, http://www.tbs-sct.gc.ca/clf2-nsi2/index-eng.asp

[39] Macromedia, 2004, Authorware 7: Courseware for e-learning, retrieved June 29, 2009

http://www.adobe.com/resources/elearning/article/lo_packager01/

[40] Nichani, Maish. "LCMS = LMS + CMS [RLOs] - How does this effect the learner? The instructional designer?" elearningpost. http://www.elearningpost.com/articles/archives/lcms_lms_cms_rlos/

[41] M-A. D. Storey, J. Bavelas, M. Wang and B. Phillips, 2002, “The Importance of Usability Testing for Web-based Learning Tools”, Educational Technology Conference Series, May 4-5, University of Victoria, Victoria, B.C., Canada.

[42] Blackboard, 2000, http://www.blackboard.com/

[43] Dokeos, http://www.dokeos.com/

[44] Moodle, http://moodle.org/

[45] The World Wide Web Consortium (W3C) http://www.w3.org/

[46] ATutor , http://www.atutor.ca/index.php

[47] Carnap R., 1939, “Foundations of Logic and Mathematics”, International Encyclopedia of Unified Science, Vol. I, no. 3, University of Chicago Press.

[48] Munoz K., Duzer J., 2005, “ Blackboard vs. Moodle. A Comparison of Satisfaction with Online Teaching and Learning Tools” http://www.humboldt.edu/~jdv1/moodle/all.htm, retrieved June 26, 2009.

[49] Frege G., 1892, paper “Über Sinn und Bedeutung”, in Zeitschrift für Philosophie und philosophische Kritik 100: 25-50. Translation: "On Sense and Reference" in Geach and Black, 1980.

[50] Bill Wadge and M.C. Schraefel, 1992, Reducing the complexity of software configuration, Proc. 1st African Colloquium on Research in Computer Science , Yaoundé, Cameroon, pp. 85-96.

[51] Brown G., 1998, Intensional HTML2, MSc Thesis, University Victoria , Canada

[52] Swoboda P. and Wadge W.W., 1987, “ Vmake, ise and ircs General tools for the intensionalization of software systems” , Intensional Programming II, World-Scientific , In M. Gergatsoulis and P. Rondogiannis, eds.

[53] Plaice J. and Wadge W. W., 1993,“A New Approach to Version Control”, IEEE Transactions on Software Engineering, Vol. 19, No. 3, pp. 368–276.

[54] Wadge W. W., Brown G., Schraefel M.C., and Yildirim T.,1998, Intensional html, Principles of Digital Document Production,In E.V. Munson, C. Nicholas and D. Wood, eds.,. LNCS Vol. 1481, P. 128–139.

[55] Swoboda P. and Wadge W., 2000, Vmake and ISE General Tools for the Intensionalization of Software Systems, In M. Gergatsoulis and P. rondogiannis, editors, Intensional Programming II, P. 310-320

[56] Feldman S.I., 1979, MAKE – A program for maintaining software , software-practice and Experience,Vol.9, no.3, pp.255-265,

[57] Tchuente M., 1992, “Reducing the complexity of software configuration” Proc. 1st African Colloquium on Research in Computer Science, Yaoundé, Cameroon, Oct. 1992, pp. 85-96.

[58] W. Wadge, G. Brown, M. Schraefel and T. Yildirim, 1996 “Intensional HTML”, Proceedings of the 4th International Workshop on Principles of Digital Document Processing,.

[59] Yildrim T., 1997, Intensional HTML, MSc Thesis, University of Victoria , Canada

[60] Swoboda P., 1999, Practical languages for intensional programming Intensional programming ,MSc. Thesis, University of Victoria, Canada

[61] Wadge W. and Schraefel M.C., 2000, Putting the Hyper back in Hypertext, Intensional Programming II, Based on the papers at ISLIP, World Scientific Publishing Co,

APPENDIX 1: SYSTEM DESIGN

The Diversity in an Environment for Accessible Learning (DEAL) framework evolved to provide an extensible, maintainable, secure, and usable web-portal for the research project. The purpose of the website is to provide the following:

1) Organize and store research work
2) Describe work and research to the public
3) Provide a distribution platform for application releases
4) Catalog all presentations, reports, papers and documentation
5) Provide collaborative opportunities

The DEAL framework structure was designed with these main functionality purposes in mind. Future capabilities and/or modules, such as an 'online community', can easily be added to the system at a later date. The above purposes may have different levels of user access. Some examples of these different levels or user groups include guest (general public), registered user, administrator, and collaborators.

The organization, storage, and categorization of all presentations, reports, papers and documents will be handled with a file-upload and file-viewing system, available only to the administrator and potentially collaborators.

The description and distribution of any published work or research appears in the library portion of the site. The administrator has the ability to publish anything in the file-viewing page module (which was uploaded via the file-upload module) to the library, to any number of user groups. The library will be administered in a library admin module, which will allow the creation and deletion of folders and subfolders and any other library maintenance task needed.

The library will present this information to the users, allowing them to view and download anything they find relevant. The framework itself will be explained below in detail.

A1.1 Description

The DEAL framework is written in PHP, and is Object-Oriented (OO). By becoming Object-Oriented, we were able to allow for much greater extensibility. The goal was to create a solid, secure core. This core, or framework, allows any future developer to quickly create a module that the framework will run.

Should the developer require something that is not contained within the modules, they are able to extend existing components. By extending components, we ensure that the base core code is never touched, thereby making upgrades easy to install for the entire system without the worry of corrupting modifications.

The framework contains several main components. These components are:

1) Engine
2) Error
3) DB
4) Configuration
5) Module
6) Security
7) Session
8) Template

Each of these components will be explained in Components section.

A1.2 Relationship within the System – How Does System Works

This framework provides the complicated engine that will drive an easy to maintain and dynamic site. First, the engine will be described in context to the other component handlers that it controls.

Engine-

1) Instantiates Error controller- This is needed by everything and has to occur first.

2) Loads the Configuration controller- Requires reference to Error

3) Start Security Handler- Requires reference to Engine

Load the DB controller - Requires reference to Engine, and various attributes from the Configuration controller

4a) Check with DB controller if a connection is obtained and valid

5) Call Configuration's `loadConfig` routine - Requires a DB controller object

6) Initialize Security Handler- Records IP and times, and has utility functions to clean DB values, validate files, clean responses, perform a security audit, and record security breaches. Also verifies that a user should have module access with the '`hasModuleAccess`' routine, which is called in step 9.

7) Start the Session Handler- Requires reference to Engine. This checks to see if there is an existing session, and if there is, it updates the expiry time. If there is no session, one is created.

   7a) this class will load the User Handler itself. A User Controller object is actually created, which in turn verifies the existence of a guest user (the System ALWAYS needs this to exist). If guest is somehow missing, it is re-created. Otherwise the user ID is obtained from the Session, and the user is created.

8) Load Module Handler- Requires reference to Engine. This is how engine prepares individual modules for loading, checks if the module exists, and if it does it 'sets' it to the handler. To set it acquires the module ID, so that security can be confirmed in step 9.

   8a) if the module is an invalid module, the configuration value for the 404 module is loaded instead.

9) Call Security's **`hasModuleAccess`** routine. Checks the ACCESS flag developer or admin selects that the module should be restricted. (ie: view funding details or auditing pages should only be accessible to the funding providers).

   If the flag request fails for some reason, the **`module_metadata`** has the ACCESS level for it as well, declared as 'ACCESS_DEFAULT'. In other words, if a table driven ACCESS request fails, the Security asks the module in question for the ACCESS_DEFAULT data from the module metadata.

   9a) determines if User should have access to the requested Module. If not, the specified unauthorized module is loaded.

10) Start the Template Hander - Requires reference to Engine. The template handler gives module a template object. Module then instructs the template object to set variable1, set variable2, and so on. When all variables are set in the template, module saves the template. Module then tells the template handler to save this template to the final version. The final version is an html file representing what the final template looks like.

11) Pass metadata to the Template Handler - Engine asks template controller for the final template, takes the data, wraps the headers around it (the global template) and returns a string.

12) Display Final Template - Displays the populated global template to the screen.

Summary of how it Works

Every page on the website that the user sees is the framework's engine presenting the user with a page module (template and module logic). As we explain in the components section explain, the reasons are security and extensibility.

When a user points their browser at the DEAL URL, they will immediately be directed to the DEAL homepage. At this point, behind the scenes, a Session object has been created, and is treating the user object (which is a sub-object of session) as 'guest'. By designing the system as an engine, which handles security through session, user, and user group objects, which is all data driven, software can easily control which pages are accessible to which end user, and or user group.

All page redirects, reloads, or query parameter submissions are handled through Engine, maintaining the security aspect of the design. Any post variable sent can be checked, and security can be enforced throughout.

Multiple page modules (and their corresponding templates) exist for the site, and new ones can be implemented at any time. To implement a module, the developer must:

1) Write the module and place it in
   `/framework/lib/class/modules`
   (this is more of a best practices than a requirement).
2) Write the template(s) and place it in
   `/framework/templates/default/global/<folder name>`
   where folder name is a folder created to hold all the templates required for the module in question.
3) Declare the module to the system – Insert relevant data **`(module id, name, description)`** into the database table 'modules'
4) Declare module access via the database table **`'module_access`**'. Default is 'ACCESS', but can be user and or group based. The Default is a configurable value.
5) Declare **`module_metadata`** via the database table '**`module_metadata`**'.

Some page modules that exist already are:

Module name	**function**
mod_userprofile	Creates a user ability profile, called when user created
mod_register	New users created here
mod_partners	Description and links of Partners / Sponsors
mod_login	Users can login to DEAL system here.
mmain	Main page / webpage’s Home page
m404	404 page

ADMIN MODULES	These are located in **module/admin**
admin_main	Administrator log in
admin_unauthorized	Administrator unauthorized page
m404	404 page

Errors

- The system logs errors, based on level. These can be found in the directory **`/var/www/html/framework/logs/`**
- There are three default levels of “errors”. They are: debug, error and trace. These levels are configurable from the error configuration file located in: **`/var/www/html/framework/includes/`**
- The usual apache log file located **`/vars/log/httpd/error_log`** can also be useful.

A1.3 Components

Each of the 8 main components is defined below:

A1.3.1 Engine

This part of the system is responsible for maintaining the objects of the other handlers. It also creates handlers and performs basic logic for the flow through these handlers. It works with the module, to assist the module in providing a powerful and working website.

A1.3.2 Error

This component is very important. It overrides the default PHP error handlers and routines. This allows the developers to capture errors or warnings with different levels for various reasons.

Name	Description
`reportError ($level,$errorCode,$errorName)`	Verify level is acceptable, create ErrorItem and queue it. Echo error to screen if required. If error triggers queue, run the queue. Returns: true or false
`handleError($errno,$errstr, $errfile,$errline)`	Handles an error, resolves $errno and sets level. This method is the override of the default PHP error handler. Calls: `reportError($level,$errorCode,$errorName)` Returns: void
`handleException($e)`	This method is the override of the default exception handler. Calls: `reportError($level,$errorCode,$errorName)` Returns: void

A1.3.3 DB

The database handler loads a connection to a database, and provides simple access to query the database. The hander supports ONLY *parameterized execution*.

The reason for allowing only parameterized execution is so that the engine can control objects sent to it for replacement and ensure they are clean. It does not look for bad data, but simply performs measures to prevent the bad data.

The handler returns a `resultSet` object (or the # of rows affected, for INSERTS/UPDATES etc). The `resultSet` object allows for iteration and or selection of data, and can return a count of how many items it has.

Queries run against the database should conform to the SQL or SQL92 standard, and should not become too specific to any single database technology. (ie: do not limit queries using LIMIT(mySQL) or ROWNUM(Oracle)). This makes the DEAL framework even more extensible.

The abstraction of data access away from the front end user is CRITICALLY important to any website or framework. By only allowing parameterized queries, we are guarding against undesirable variables being potentially passed in.

Name	Description
`prepareStatement($inSQL)`	Creates a PreparedStatement object. Returns: Reference to the object
`executeStatement([$inStatement])`	Executes supplied PreparedStatement Return: ResultSet object
`executeNonQueryStatement([$inStatement])`	Executes supplied PreparedStatement Return: int - # of effected rows
`getLastInsertId()`	Return: int – the last inserted ID
`getError()`	Return: string – the last error message
`getErrNo()`	Return: int – the last error number.

A1.3.4 Configuration

This part is described as System/Module level configuration meta-data.

The configuration component reads simple information from a configuration.php file (located in the **`/includes`** directory). The rest of the data comes from the database. The configuration manager provides basic functionality and allows the developer to retrieve, set (for the runtime), save (permanently) and utilize the configuration data.

Name	**Description**
`get($requestedKey)`	Provides the value for specified key. Returns: null or invalid or value
`set($requestedKey,$newValue)`	Set specified key with specified value. Returns: true/false
`exists($requestedKey)`	Checks if a key exists. Returns: true/false
`save($requestedKey)`	Checks if key exists, then re-saves value to new key. Returns: true/false
`rollback($requestedKey)`	Rollsback a config object. Returns: true/false

A1.3.5 Module

The module handler component plays a major role in the framework. It controls the loading, reloading, and unloading of modules from the system. The module handler is responsible for ensuring the requested module is valid and available. The module handler then performs following actions:

1. It will validate that the requested module exists. If it does not, the ‘404 Not Found’ error module will be loaded instead. The Module handler determines the module to run by checking the “act” variable from POST and then GET. This variable determines which module is loaded. The module is validated to exist by checking that the module (a) exists within the database and (b) exists as a module class.
2. The module’s metadata is loaded in preparation for next step, and to determine that module is still valid.
3. Once it is determined that module is valid, it will load a global class that is responsible for ensuring:
    a. user has access to the module
    b. module is loading properly

    c. it runs any actions specified by the module's metadata

4. The module determines which method it runs. By default, the "**`execute()`**" method of the module class will run. Other *public* classes of the module can be requested to run by specifying the "do" POST or GET variable. If the method requested does not exist, **`execute()`** will be chosen. This allows the module to dynamically call different components within its layer.

As touched on above, there are 2 distinct parts to any module.

1) Module Metadata - Info contained here are: **`module_name`**, **`version`**, **`last_update`**, and use of the global templates. Metadata must ONLY contain information about the module, NEVER primary business logic. That goes in the module business logic!

2) Module business logic – This is the runtime of the module. It is basically an application that utilizes the API's provided by this framework. By the time it reaches this point, the system has already recognized that the user is authorized to view the module, and the module metadata has confirmed this action as well. At this point, the module should not be redirecting users to a '**404 Not Found'** or '**Unauthorized Access'** modules. It can, however, limit what the user can see based on the securities module. Also, redirecting to a new module is allowed, should the logic of the module determine that it is needed. (ie: data missing from input, so user is redirected back to search screen).

All Modules implemented must extend **`ModuleObject,`** and override **`execute()`**. The execute() routine has the module object create any forms and templates needed, and sets required variables. Then template is then saved.

Setting variables requires two parameters: the variable **`placeholder`** name, and **`value`** to be substituted in.

Any template for the DEAL framework will not require html form tags. When a form is required in a template, the module object tells the template object to create a form, which takes as a parameter the name of a module object's routine (ie: function login() in the module object would , which will be called the form's action/input type="submit").

All html input tags that are children to form tags can be used by the designer. However, as described above, the template is completely view layer, and the declaration, and placement of the routine which is called by the form, resides in the module object that calls the template.

This design approach abstracts the always security-threatening form, and all necessary security around it, away from developers and designers. Form logic is handled by the template controller. This sets query variables and the module name so that on form submit, the engine is recalled which reloads this module object's routine to handle the form submit, passing to it the post variables (which have been cleansed and secured by Security, after they were passed to Engine).

Name	Description
`getModuleName()`	Returns: module name
`getModuleID()`	Returns: module id
`executeModule()`	Calls modules execute() routine Returns: void
`loadUnauthorized()`	Confirms unauthorized module exists, and if so calls resetAndReload($inModule) Returns: void
`resetAndReload($inModule)`	Unsets all old data, confirms module exists, and resets any templates. Returns: void

A1.3.6 Security

The security component handles the controlling of access to modules and system level functions within the system, based on a unique identifier for a user. The security handler will always be in place, even if a User database is not in use. This handler operates under the assumption that everyone has a user account. If no user account exists, the user is considered a "Guest" account.

****NOTE**: The above described 'no user' check actually is handled by the User component, which is handled by the Session handler.

The security handler also ties into the DB, Session, and User modules, as they all pass data from the user.

Based on the level of auditing selected, the securities handler will record the information into the database or a flat file. The information is keyed based on the **`sessionID`**, **`user ID`**, **`user's IP`**, and any unique identifiers.

The security handler is a VERY LARGE part of the system. Aside from the above-described handling of auditing and user access, it also provides methods that a developer can use to:

(i) Sanitize Queries

(ii) Ensure valid data has been passed, such as emails or URL's

(iii) Prevent against cross-site-scripting

(iv) Prevent email injection and more.

Additionally, the Security handler is also configured to monitor all passed data for attempts at exploiting the PHP (i.e.: null and non-chars being passed), and to immediately reject requests and log such incidents when any of the above mentioned issues are detected.

Future Additions will see the security component handle auditing of data. Data auditing is yet to be determined but will likely be triggered if submitted data is suspected to have attempted XSS or SQL Injection. This would be stored in a separate table.

Name	Description
`initialize()`	Lets security know when database is ready. Returns: void
`cleanDBValue($inValue)`	Cleans a value while preventing SQL injection. Replaces common bad chars. Returns: cleaned value
`validateFile($inFile)`	Prevents file injection. Ensures name provided doesn't contain funny chars. Also ensures isn't trying to load inappropriate files (i.e. `/etc/passwd`) Returns: $inFile
`get($key)`	Prevents against null byte Returns: null or the cleaned post value
`post($key)`	Prevents against null byte Returns: null or the cleaned post value
`hasModuleAccess()`	Checks module ACCESS flag from database, based on user id and (if relevant) group id. If no flag exists in module_access table, the module metadata is checked. Returns: true / false

A1.3.7 Session

The Session handler is limited to handling information about an actor's current session state. The session handler does not contain any login logic, nor does it handle user information logic. Its sub-object 'User' handles this information.

By default, this sub-object will handle basic login, logout, and user data of the specific user. Custom login options (like vBulletin or LDAP) could be used to extend the base module and update the business logic within the extended object.

The Session component/handler can be though of as a configuration module, but PER USER and based on an active session state. The session module is configurable through the configuration controller. The important configurable options are: use of cookies, and session length.

Name	Description
`startUser()`	Creates a UserController object Returns: void
`get($inKey,$foundData)`	Gets specified session key value from database Returns: null or value
`set($inKey, $value)`	Sets specified session key-value pair. If key exists, value is updated, otherwise a new key-value pair is created Returns: true / false

A1.3.8 Template

Template controller is controller for view layer (the html). It allows the module to add templates and generate dynamic content into a document that can be outputted to the site. It separates the model and controller portions of the system from the view, providing a clean playground for designers and a separate code base for developers.

The templates, generally HTML files, are stored within a templates folder. Based on the TYPE of site, defined within the main page, the website utilizes a pseudo template sets to display the site. For instance, a TYPE of *admin* will be stored in **`admin_global`**, while the main site is stored in *global*.

These template files use special tags. The template engine is a prototype engine based on the *Krownet Template* system. It uses the *KT** tags: <KTVariable> and <KTBlock> to generate the final output.

When a template is first called, its template file is scanned and a hierarchical representation of the file and its components are represented within memory using arrays and objects. The module can then call upon the template controller to return a "template" object for itself. The template object, once called, is immediately added to the display queue.

A template object can have its variables set, and it can retrieve “KTBlocks”. A KTBlock is a section of the requested document, wrapped in a <KTBlock></KTBlock> tag. The sub block can do the same as template, and can even return sub-sub blocks, to an almost-infinite level (limited by memory).

The following is an example of a template file:

```
<b><KTVariable name="foo" value="Default value" /></b>
<ul>
<KTBlock><li><KTVariable name="SubObject" /></li></KTBlock>
</ul>
```

The above HTML file uses both types of KT tags. The definition of the tags is as follows:

KTVariable:

Attribute	Description
`name`	The descriptive name of the object. This is used to set its value within the module.
`value`	The default value of the variable for regular variables. Value may have other purposes for various types.
`type`	The type of variable is important. There are various types available. By default a "variable" type exists and a "system" type exists. System are system level parameters. Variables are standard variables that are set and display data. Other variables can be added to the ***`lib/class/objects`*** folder. The file and class name must extend a VariableObject and follow the name convention of *Type**Variable*** (type value suffixed with Variable). Some examples of Variable types are Block and *TemplateDir*. *TemplateDir* returns the specified value, prefixed with the Template Directory. Block, will call a block_*value* object from the *lib/class/blocks* directory. The block will return a string that the template will replace the KTVariable with.

KTBlock:

Attribute	Description
`name`	The descriptive name of the object. This is used by the module to retrieve a template object for this block of display code.
`separator`	Anything with this tag will suffix the end of each block statement except the last one. Useful when separating by a horizontal line that you don't want the last one to be separated by.

Using each of this variables, we are able to generate dynamic web pages with a complete separation of code and display. The reasoning for this, is that designers can now design a full website without having to know PHP to be able to work. It also means that developers don't need to modify the design to make it work with their system.

Template Controller:

Name	Description
`getTemplate($templateName)`	Returns a template object.
`getTemplateObject( $templateName, $templateFile)`	Returns a template object without adding it to the template queue.
`newTemplate($templateName, $templatefile)`	Returns true if template is found. Adds the template to the display queue.

Template Object:

Name	Description
`getBlock($blockName)`	Returns an array. An array is returned because it contains metadata necessary for saving the block. Using $returnedArray['template'] will gain access to a template object for the array.
`save()`	Save the changes made to the array. Returns true. This is used especially on blocks. Saves the chosen variables and allows the developer to run through it again and have the view appear twice on screen.
`finishBlock($blockArray)`	Finishes the block. Once the block is finished it cannot have another iteration.
`setVariable ($varName, $varValue)`	Sets the value of the variable. If variable does not exist, it acts like its set. The variable existing or not means nothing. If a variable is removed from the design that's the designers choice, and thus why its separated into the view.

A1.4 Data Requirements

A1.4.1 Data Design

This is the frameworks data storage structure, as of August 20th, 2008. It is inevitable that more tables will be added to the system, to facilitate the creation and support of new modules at a later date. Our development is not yet complete on the core framework as of the date of this documents creation, August 20th 2008, and hence some the existing tables described below could possibly see a structure change.

This is a fully relational database, with normalized tables in third normal form. Primary keys must exist for every table, and for most are sequence generated.

//TODO do ERD of tables when they are all there. No ERD is available at this time.

A1.4.2 Data Definition

TABLE: `< config >`

Column Name	Type	Description
cid	Int(11)	Primary key. Sequence generated
key	Varchar(255)	Key value pairs – key
value	Text	Key value paris – value

TABLE: `< component_access >`

Column Name	Type	Description
cid	Int(11)	Primary key. Sequence generated
iid	Int(11)	The item ID, based on type.
uid	Int(11)	The user ID this is for.
gid	Int(11)	The group ID this is for.
type	Varchar(255)	The type of module this is for.
flag	Varchar(255)	The flag is either to grant ACCESS or DENY

TABLE: < `groups` >

Column Name	Type	Description
gid	Int(11)	Primary key. Sequence generated
code_name	Varchar(255)	
name	Varchar(255)	
description	Varchar(255)	
base	Char(1)	

TABLE: < `modules` >

Column Name	Type	Description
mid	Int(11)	Primarky key. Sequence generated
code_name	Varchar(255)	
Name	Varchar(255)	

TABLE: < `module_access` >

Column Name	Type	Description
maid	Int(11)	Primary key. Sequence generated
mid	Int(11)	fk to module table pk
gid	Int(11)	Default is NULL. fk to groups table pk
uid	Int(11)	Default is NULL fk to users table pk
flag	Varchar(8)	Default is 'ACCESS'

TABLE: < `module_metadata` >

Column Name	Type	Description
mmid	Int(11)	Primary key. Sequence generated
mid	Int(11)	fk to module table pk
key	Varchar(255)	Key value pair – key
value	Varchar(255)	Key value pair - value

TABLE: < `security_audit` >

Column Name	Type	Description
said	Int(11)	Primary key. Sequence generated
occured	Datetime	
Uid	Int(11)	
mid	Int(11)	
get_dump	Text	
post_dump	Text	
engine_state	Text	
ip_address	Varchar(255)	

TABLE: < `sessions` >

Column Name	Type	Description
sid_seq	Int(11)	Primary key. Sequence generated
sid	Varchar(255)	
ip_address	Varchar(50)	Ip of who session belongs too
session_start	Int(11)	Session start time
session_expire	Int(11)	Session expiry time

TABLE: < `session_data` >

Column Name	Type	Description
sdid	Int(11)	Primary key. Sequence generated
sid	Varchar(255)	
key	Varchar(255)	Key value pair – key
value	Varchar(255)	Key value pair - value

TABLE: `< users >`

Column Name	Type	Description
uid	Int(11)	Primary key. Sequence generated
username	Varchar(255)	
password	Varchar(255)	
last_login	Datetime	
login_attempts	Int(11)	Default is 0
last_login_fail	Datetime	
guest_flag	Char(1)	Default is 0

TABLE: `< user_data >`

Column Name	Type	Description
udid	Int(11)	Primary key. Sequence generated
uid	Int(11)	fk to users table pk
key	Varchar(255)	Key value pair – key
value	Varchar(255)	Key value pair - value

TABLE: `< user_group >`

Column Name	Type	Description
guid	Int(11)	Primary key. Sequence generated
uid	Int(11)	fk to users table pk
gid	Int(11)	fk to groups table pk
priority	Int(11)	

TABLE: < `user_profile` >

Column Name	**Type**	**Description**
upid	Int(11)	Primary key. Sequence generated
uid	Int(11)	fk to users table pk
impairment_visual	Int(1)	Treat as boolean. 1=yes. Default=0
impairment_hearing	Int(1)	Treat as boolean. 1=yes. Default=0
impairment_pysical	Int(1)	Treat as boolean. 1=yes. Default=0

APPENDIX 2: Markup Macro Processor (MMP)

A2.1 Definition

MMP is a general purpose Macro Processor in the tradition of TRAC, M6, the troff/groff macro facility and others. It accepts an input text document and produces a corresponding output text document which results from copying ordinary text unchanged while expanding macro calls.

MMP differs from other macro processors in the form that macro calls take: they look like XML elements, using a slightly changed and vastly simpler syntax. Expanding MMP macros can therefore be thought of as implementing user-defined tags.

MMP differs from XML in that tags are marked using curly parentheses ({ and }) rather than angle brackets. Also, attributes are not named; instead, the order in which they appear determines their signiificance (for which reason they are called *arguments*).

Here is a typical MMP macro call:

```
{memo Tom Dick}I received your proposal.{/memo}
```

This is a call to a macro called memo. The first argument (actual parameter) is Tom and the second argument is Dick. The text between the opening and closing tags, I received your proposal, is called the *body* of the call.

Once the MMP processor has recognized and parsed a macro call, it replaces the entire call its *value* or *result*. Typically (but not always) this value is defined by a template into which the arguments and body are substituted. For example, the above call might produce

MEMO

From: **Tom**

To: **Dick**

I received your proposal.

Notice that the result has newlines in it – MMP is not line oriented, and newlines play no special role (other than as whitespace – see later). Templates can have embedded newlines as can bodies and even arguments.

Even though the MMP calling format is based on the XML syntax, there is no possibility of confusing the two – angle brackets have no special status at all for MMP. This makes it easy to define macros that produce XML markup. In fact that is one of the major motivations.

As with XML, MMP allows calls consisting of a single tag and no body. For example, the call

```
{dept Physics/}
```

might expand to

> `Physics` Department of the University of Victoria

Although single tag calls have no bodies, the can still have arguments, as in the example.

Evaluation

MMP macro calls, like XML elements, can be nested. For example,

```
{memo Tom Dick}I received a spreadsheet from the {dept Physics/}.
{/memo}
```

Whenever calls are nested, the question arises, in what order are the calls evaluated. In the example we could expand the call of "dept" first, so that the call to "memo" would have

> I received a spreadsheet from the `Physics` Department of the University of Victoria

as its body. Alternatively, we could first expand the "memo" macro with

> I received a spreadsheet from the `{dept Physics/}.`

as its body, then expand any occurrence of dept in the resulting text.

In this example, the end result will be the same, and this is often the case. Sometimes, however, it will make a difference. MMP avoids any possible ambiguity by *always evaluating inner* *macros first*. Inner-first evaluation happens even if (as is possible) the enclosing macro ignores its body, and therefore ends up discarding the result of the inner evaluation.

Quoting

All macro processors must have some mechanism to force text to be handles verbatim, with macro expansion turned off. This is necessary if text may accidentally contain patterns that MMP would otherwise interpret as (parts of) a macro call. Also, sometimes we need to override the inner-first evaluation rule and delay evaluation until later.

MMP provides a special macro for this purpose. Its name is " (a double quote character) and it is called like any other macro; for example

```
{"}Greetings from the {dept Physics/}, everyone.{/"}
```

The quote macro returns its body as its result. However, the body is *not* evaluated first; it is passed on verbatim. The quote macro is the *only* macro with this property. No other built-in macro does this, and it is not possible for users to define their own macros that quote their bodies.

Of course, quoting is not foolproof. If the text we are quoting itself contains the sequence {/"}, MMP will be fooled. If this is what we want, we must be ingenious (more later).

It’s actually not unusual to want to quote the body of a macro call. As we just said, we cannot define a macro that does this automatically. We simply have to enclose the body in a quote macro:

```
{memo Tom Dick}
{“}I received a spreadsheet from the {dept Physics/}.{/”}
{/memo}
```

This can be really cumbersome, so MMP allows quote to be used as a kind of meta-macro:

```
{" memo Tom Dick}I received a spreadsheet from the {dept Physics/}.
{/"}
```

Here quote is being called with three arguments and a body. Calls to quote like this are evaluated as a call to the first argument (treated as a macro name) together with the remaining arguments as the arguments of the implied call. The body of the implied call is the verbatim (quoted) body of the original call. Notice that this is a call to the quote macro (not the memo macro) so that the closing tag is {/"}, not {/memo} (an easy mistake to make).

Macro Definitions

Most macros are defined by substitution templates, each template being a string with ‘holes’ for the arguments and the bodies. In MMP these holes are special ‘system’ macros. The macro calls that act as placeholders for the arguments are

```
{1/}  {2/}  {3/}  …
```

In other words, a call to the macro 1 without arguments or body marks the position where the first argument is to be substituted. Similar calls to macros “2”, “3”, “4” and so on mark substitution points for the second, third, fourth etc arguments. The call {_/} to the macro “_” marks where the body is to be substituted.

The definition of an ordinary template macro is therefore simply a string. Users can define their own template macros using a special system macro “:”. This macro is called with one argument and a body. Evaluating such a call to : returns the empty string but has a side effect: it enters the body as the template string for the (macro whose name is) the first argument.

For example, evaluation of the call **`{: uvic}University of Victoria{/:}`** associates the template string University of Victoria with the macro name uvic. Evaluation of the call {uvic/} thereafter expands to University of Victoria. The uvic macro can be called with arguments and a body, but they will be ignored, because none of the special system macros just discussed appear in the template for uvic. For example,

```
{uvic Tom Dick}are students at the{/uvic}
```

will also expand to University of Victoria.

Suppose instead we wanted the template for uvic to be

```
{1/} and {2/} {_/} University of Victoria
```

so that the last call would expand to

Tom and Dick are students at the University of Victoria.

We might rush to the keyboard and enter the following call to:

```
{: uvic}{1/} and {2/} {_/} University of Victoria{/:}
```

But, this will blow up in our face. As we already explained, apart from every MMP macro evaluates its body first. This means that the MMP processor will attempt to evaluate {1/} – and fail since there is no complete macro call ready to be evaluated, and hence no first argument. (Nor is there yet a second argument or a body).

Clearly, we want: to take the string {1/} verbatim, without expanding it. So we need to quote the body of the call. The direct way to do this is

```
{: uvic}{"}{1/} and {2/} {_/} University of Victoria{/"}{/:}
```

This works fine but is a bit clumsy. The idomatic way to do this is with a call to " with : and uvic as arguments:

```
{" : uvic}{1/} and {2/} {_/} University of Victoria{/"}
```

Quotes are also needed if you want to the template of a macro to contain a nested call meant to be expanded when the macro you are defining is itself called.

For example, suppose you want the macro greeting to have the template

Welcome to the Uvic class of `{thisyear/}`.

The call

```
{" : greeting}Welcome to the Uvic class of {thisyear/}.{/"}
```

does the job. If later on you add `{: this year}2007{/:}`, the call {greeting/} will expand to

Welcome to the Uvic class of 2007.

Suppose, however, by mistake you use the call

```
{: greeting}Welcome to the Uvic class of {thisyear/}.{/:}
```

The difference is that MMP will now try to expand the call **`{thisyear/}`**, and then use the result as part of the template to be associated with greeting. If you haven't already defined the macro **`thisyear,`** you'll get an undefined macro error. If you have already defined **`thisyear`** to be 2007, everything works fine – at first.

Problems arise, however, in the New Year. In January you redefine thisyear to be 2008; but when **`{greeting/}`** is called, you get the message from 2007. On the other hand, if you'd used the quoted definition for greeting, it would expand in January to

Welcome to the Uvic class of 2008.

Parsing the Argument List

As we already indicated, macro calls can have arguments, correpsonding to the attributes of XML. MMP arguments, however, are much simpler. The arguments (if any) are listed in the tag following the macro name, separated from it and each other by blanks or other whitespace characters (newlines are treated as whitespace). Arguments are ordered, not named, and can be arbitrary strings. Arguments can optionally be enclosed in double quotes, and these quotes are necessary for arguments that contain whitespace or any of the characters {, }, or /. Quote characters can also appear in arguments, in which case they must be escaped with \.

The following tag illustrates these rules (for the sake of precision the example is not indented):

```
{fred @yg4 "tom" "dick and jane" alpha beta
"long time no see" "$%^\"*"/}
```

In this example **`fred`** is called with seven arguments. The first five arguments are (one per line, no indentation):

```
@yg4
tom
dick and jane
alpha
beta
```

The sixth argument,

```
long time
no see
```

has an embedded newline, and the seventh,

```
    $%^"*
```

has an embedded quote.

A macro call can have an arbitrary number of arguments, independent of its definition. If a template macro is called with more arguments than are referred to in its template, the extra ones are ignored. If not enough are supplied, an evaluation-time error will result. The only syntax checking done by MMP is to ensure that individual tags are well formed, and that opening and closing tags match (have the same name – are properly nested). There are no MMP DTDs.

Evaluation of tags

We've already seen that the MMP processor evaluates the body of a macro before it evaluates the macro itself. The same is true of the tags that form the macro call, though we have yet to see an example.

Suppose that Tom wants to send a thank-you memo to Brad Steel, Dean of Engineering. Using the memo macro defined above, he can write

```
    {memo Tom "Brad Steel"}Thank you for the visit.{/memo}
```

But suppose Tom defines a macro for the Dean of Engineering:

```
    {: DE}Brad Steel{/:}
```

Can Tom use this macro for the argument of the memo macro? Yes, he simply writes

```
    {memo Tom "{DE/}"}Thank you for the visit.{/memo}
```

The MMP processor will encounter the call to the DE macro while parsing the opening tag of the call to memo. When is does, it will suspend the parsing and evaluate the call, then resume the parsing, consuming initially the characters produced by this macro expansion.

Notice that the quote symbols do *not* disable the expansion of the call to DE. MMP distinguishes between quote symbols and a call to the quote macro. A call to the quote macro will disable expansions in its body. However, quote symbols serve only to prevent embedded whitespace from acting as an argument separator.

If Tom does not enclose the call to DE in quote symbols, as follows

```
{memo Tom {DE/}}Thank you for the visit.{/memo}
```

Then, expanding the call to DE results in

```
{memo Tom Brad Steel}Thank you for the visit.{/memo}
```

In this call the second argument is Brad and the third argument (which memo ignores) is Steel.

In this case the result is probably not what Tom wants, but sometimes this phenomenon is useful; it allows us invoke a whole series of arguments with a macro call. Suppose that Tom's invite macro generates a series of invitations, one for each argument. Then the call

```
{invite Dick Harry Pam Sally Rajiv/}
```

will generate an invitation to each of Tom's five best friends. If he wants, Tom can define a macro pals by

```
{: pals}Dick Harry Pam Sally Rajiv{/:}
```

and then invite them with

```
{invite {pals/}/}
```

Macros can of course be body- or argument- nested to any depth.

Macro Processing – Call Parsing

It should be clear that with quoting and nesting, the evaluation process is not as simple as it might appear. Therefore we give a somewhat more detailed description of the evaluation algorithm.

As we already explained, MMP (like most macro processors) passes input directly to output until it encounters a macro call. So the real processing starts when MMP finds `{` in its input stream.

When it does, it assumes that the open curly is the first character of an opening or opening/closing tag. It therefore parses the tag, and consumes all the characters that make up the tag (meaning, does not pass them on to the output). The parsing process is simple, because the syntax of tags is simple: an opening curly paren, a name, and then a possibly empty sequence of arguments, each preceded by at least one whitespace character. Names and arguments are strings of characters none of which are whitespace or any of the special characters `{, }`, **and** `/.` Finally, a tag is terminated by either `}` or `/}`.

Names can have `"` embedded but not arguments. However, arguments can be enclosed in "s, in which case whitespace characters are taken as part of the arguments. Furthermore, the special characters and `"` can be part of an argument string if they are escaped by `\ (\` itself can be escaped).

If the tag parser encounters an unescaped `{` (even within a quoted argument), it assumes that it has come across a macro call nested within the tag. It therefores suspends parsing until the call has been evaluated, then resumes parsing at the first character produced by the evaluation. It does *not* evaluate macro calls appearing in these characters. Another way to describe this is that the processor gathers the characters that make up the tag, evaluating where necessary, then parses these characters without further evaluation.

Once the tag has been evaluated, what happens next depends on whether it is an opening or opening/closing tag (whether it ends in} or in `/}`).

In the latter case, the processor takes the name and arguments and assembles them into a macro *call* object. These objects wrap up all the data needed to evaluate a call: the name, the arguments, and the body.

In the case of a open/closing tag there is no body, but there is a name and arguments. This call object is not stored, but passed on directly for immediate evaluation (see below). The results of the evaluation are placed at the head of the input stream, and the processing continues.

If the tag is an opening tag, we again assemble the name and arguments into a call object, but we leave a slot open for the body. This *incomplete* call object is placed on the ‘incomplete’ stack, a stack of incomplete calls whose closing tags have not yet been encountered. We need a stack because macro calls can be arbitrarily body-nested.

Once we stack the incomplete call, we proceed through the body, still evaluating. But we direct the output stream to a separate buffer so as not to mix up data output before we encountered the opening tag (this data will not be part of the body).

Once we have encountered an opening tag, we have to be on the lookout for closing tags, which begin `{/`. These are parsed according to similar rules (they can even have arguments, as explained later). Once we have parsed the closing tag, we assemble the parts (name plus arguments) into an incomplete closing tag. This closing call should be compatible with the call on the top of the incomplete stack (and will be if the calls were correctly nested).

Once we match opening and closing tags we merge them into a compete call, by combining arguments and adding the contents of the output buffer as the body. Once again, we are ready to evaluate a macro call.

Macro Processing – Evaluation

Macro evaluation is triggered whenever we have a complete call at hand. This can happen imediately, after parsing an open/closing tag, or later, when we finally parse and match a closing tag with a previously parsed and stacked opening tag. In either case we have at hand all the information needed to expand a macro: the name, the arguments, and the body (if there is one).

In either case an object encapsulating all this information is placed on the top of the 'completed' stack and evaluation proceeds. During evaluation the completed call remains on the top of the completed stack and the body, name, and arguments are all available. Of course, if evaluation results in another macro call being parsed, the new call will temporarily cover the older one until the evaluation of the newer one is complete.

The MMP processors first step is to check if the name is that of one of the built-in or system macro. If it is, then that special code is executed.

If it is not a built-in macro – if it is user defined – then the processor looks up its definition. This input is copied into the input stream in front of the input currently waiting to be processed. Conceptually, there is a marker placed between the characters of the definition and the rest of the input. Then normal input processing resumes. This step corresponds to entering the body of a subroutine in a conventional procedural language.

The processor refers to the call on top of the competed stack if it encounters any of the special system macros while processing the definition. For example, if it parses and evaluated the call

`{3/}` it retrieves the third argument stored in this call and copies these characters into the input stream.

When the processor encounters the marker which indicates the end of a copied definition, it removes the marker but also pops the completed stack. This corresponds to exiting the body of a subroutine in a traditional procedural language.

Useful System Macros

MMP provides a number of built-in or system macros that are executed for their side effects and/or produce results that cannot be obtained from a template. We have already introduced two of them, namely the definition macro : and the quote macro ". Also, the arguments of a macro call are referred to in its definition by the system macros `1, 2, 3, 4, …,` and its body by the macro `_`.

There are as well more mundane macros for carrying out humble calculations. The plus macro + sums returns the sum of its arguments (which it expects to be numerals). Thus

```
{+ 4 8 3 0/}
```

will evaluate to 15. The `<` macro determines whether its first argument is less than its second argument, returning 1 if it is and 0 otherwise (1 and 0 play the role of 'true' and 'false' respectively in MMP). In the same way `<` tests for equality.

A2.2 Macro Versions

A major innovation of MMP is allowing for macros to have multiple versions. We accomplish this by allowing multiple definitions of the same macro, each having a different tag attached to it. This tag corresponds to the context under which that particular version of the macro is to be expanded. The tag is actually an argument of the definition macro, which follows the macro's name. Consider the following example:

```
{: ferd <green> <blue>}green and blue{/:}
```

```
{: ferd <red>}red{/:}
{: ferd}vanilla{/:}
```

Here, we define the macro ferd, which has three versions. If the context under which ferd is expanded is <green> or <blue>, then ferd evaluates to green and blue. If the context is <red>, then ferd evaluates to red. Under all other possible contexts ferd evaluates to vanilla. The latter version of ferd is called the *vanilla* version, which in effect is the default version of the macro. We define the vanilla version by simply omitting the context tag or by using an empty context tag, <>. Note that the vanilla version of a macro is not mandatory.

Context Tags

We use *version expressions* to denote context. It is these version expressions that act as the tags in our macro definitions. At the ground level we have *dimension labels* and *dimension values* that come in pairs separated by a colon, `:`, e.g. `d:v` says that dimension *d* has the value *v*. The version expressions allow for nested dimensions grounded by a value, e.g. `d:e:v` states that dimension *d* has value represented by the version expression *e*:*v*.

The version expressions are elements of a *version algebra* that has a join operation, denoted +. We use the join operation to combine two versions together to make a new version, provided the two versions are compatible. So, the expression `d:v + e:w` defines the version with two dimensions *d* and *e* with values *v* and *w*, respectively.

The nesting of dimension labels in conjunction with the join operation allows a version to have a single unlabelled value, called the *base value*. Dimension labels distribute over + as follows:

```
d1:<v1 + d2:V2 + … > = d1:v1 + d1:d2:V2 + … ,
```

where d_i are dimension labels, v_1 is the base value and V_2 is a version expression.

Context Macros

The MMP interpreter determines the version of the macro to expand under the context in which it is evaluated, or the *current context*. The current context is a background phenomenon that guides macro evaluation, also represented by a version expression. There are special system macros for setting and modifying the current context.

The **[]** (or **!=**) macro allows you to change the current context. The first argument of [] is the new current context. This macro can change the current context in one of two ways; you can replace the entire current context with a new context by enclosing the first argument in angle brackets, < >, or you can modify particular dimensions of the current context by enclosing the argument in square brackets, [ ].

So, to set the current context to `<green+a:1>` we use the call `{[] <green+a:1>/}.` Then, to modify the value of dimension a to 2, we use the call `{[] [a:2]/},` changing the current context to `<green+a:2>`.

Furthermore, there exists a `!` macro that allows you to change the current context temporarily. Any macro evaluated in the body of the ! macro will be expanded under the temporary context, defined by the first argument of the ! macro. For example, the following (along with the ferd definition above),

```
{[] <>/}

before {ferd/}

{! <green+a:1>}during {ferd/}{/!}

after {ferd/}
```

will evaluate to

```
before vanilla
during green and blue
after vanilla
```

There also exists a system macro, #, which returns the value of the current context. A call to #, without arguments, returns the entire current context. To see the value of some particular dimension in the current context, you enter the name of the dimension in the first argument of the # macro. Any other arguments are ignored, (although it may be useful to allow for a number of particular dimensions to be specified). So, the following

```
{[] <a:1+b:2+c:3>/}
{#/}
{# a/}
{# a b/}
```

will produce,

```
<a:1+b:2+c:3>
1
1
```

Version Space

The interpreter's choice of an appropriate macro version is dependent on the *version space* and *best-fit algorithm*, ideas developed by Bill Wadge and John Plaice[2]. Here, Plaice and Wadge introduced version control tools for software development. In their system, different software components can have multiple versions (like MMP macros). A complete version of a piece of software is formed by taking the most relevant version of each component. The most relevant version of the components is determined by properties of the version space and best-fit algorithm.

The version space is the set of all possible contexts, partially ordered by a *refinement relation*, denoted $\subseteq$. For versions V and W, $V \subseteq W$ is read as V *refines* to W and means that W is a more specific version than V. Furthermore, we say $V \subseteq W$ *maximally* if for any version $U \neq V$ where $U \subseteq W$, then V does not refine to U.

The refinement relation is transitive, which is to say that if $V \subseteq U$ and $U \subseteq W$, then $V \subseteq W$, where V, U, and W are versions. The empty version, called the *vanilla* version and denoted <>, has no dimensions or values and refines to any version, that is $<> \subseteq V$ for all versions V. The join operator, +, is the least upper bound induced by $\subseteq$, meaning $U + V$ is the least upper bound of U and V if and only if for all W such that $U \subseteq W$ and $V \subseteq W$, then $U + V \subseteq W$. The join operation is also idempotent, commutative, associative and has the properties:

[2] Plaice, J. and Wadge, W. W. A new approach to version control. *IEEE Transactions on Software Engineering* 19(3):268-276, March 1993.

$$V \subseteq V + W,$$
$$\frac{V \subseteq V' \quad W \subseteq W'}{V + W \subseteq V' + W'}.$$

Points in the version space are the same version expressions described above. In terms of version expressions, the refinement relation is defines as

$$d_1 : V_1 + d_2 : V_2 + \Lambda \subseteq e_1 : W_1 + e_2 : W_2 + \Lambda$$

if and only if for each d_i either V_i = <> or $d_i = e_j$ and $V_i \subseteq W_j$ for some j.

A2.3 Best-fit Algorithm

The current context exists as a point in the version space. The context tags associated with our macro definitions also represent points in the version space. Using the refinement relation we compare all the versions of a macro definition to each other and to the current context and select the *best-fit* version of the rule. The best-fit version of a rule is a version that refines to the current context maximally. If there is more than one maximal version, then the system produces a failed best-fit error.

To illustrate how the best-fit algorithm works, let us consider the following example, in which we have defined five versions of an image macro to be embedded in a web page. The choice of macro is dependent on the language of the page, the type of browser and the platform of the computer.

```
{: image}<img src=”image1.gif”>{/:}
{: image <ln:en>}<img src=”image2.gif”>{/:}
{: image <ln:fr>}<img src=”image3.gif”>{/:}
{: image <ln:en+br:ie>}<img src=”image4.gif”>{/:}
{: image <ln:en+br:ff>}<img src=”image5.gif”>{/:}
```

Now, suppose that the current context is `<ln:en+br:ie+pl:pc>` and there is a call to the image macro, `{image/}`. To determine the best-fit image, the interpreter views the relationship between the five context tags and the current context as points in the version space (illustrated below with refinement relations denoted by a line).

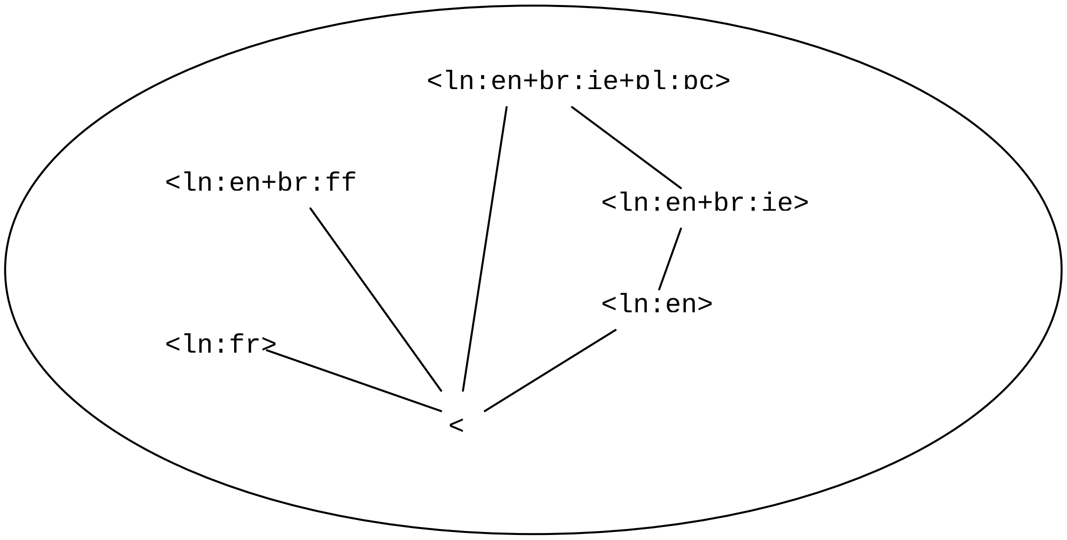


Figure A2-1: best-fit image as a Graph

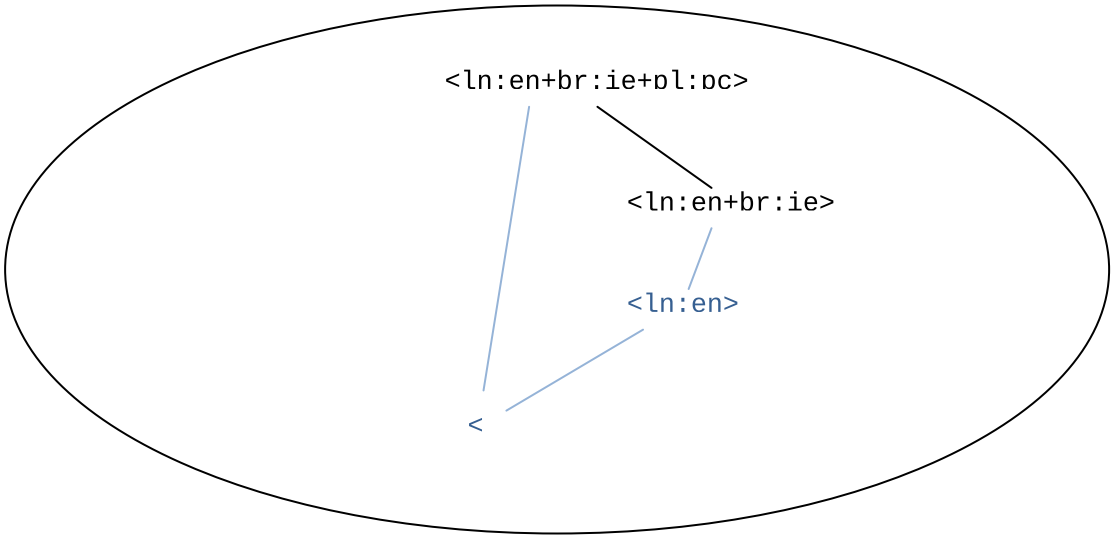


Figure A2-2: The context point the refines to the current context

Next, we throw away any context points that do not refine to the current context. Of the points that are left, if any, we choose the context point the refines to the current context maximally, **`<ln:en+br:ie>`**. So, in this example the interpreter would produce the output **`<img src=”image4.gif”>`**.

APPENDIX 3: Learner Profile

This appendix describes both the current and proposed future profile modules for the DEAL project. The profile module's purpose is to create and model an accurate **ability profile** for a user during a session using DEAL.

When a profile is complete, metadata related to the profile will be embedded in the template that the DEAL web framework has created. This is done before the template data is passed through the MMP engine. The MMP engine utilizes this profile metadata to determine the kind of learning object that would be the most useful to the user. The subsequent delivery of the custom learning object is out of scope of this profile module summary. This document also attempts to propose a path forward.

This module will use a script of questions to gauge a degree of ability across the three major ability genres (physical, vision, and hearing.) A more accurate ability profile, with a numeric representation for each genres level of ability for that user, is created.

This profile will be embedded as metadata into the template as the current profile module does. However, this metadata should now be far more accurate, and the process of acquiring the information should feel less intrusive to the user, due to the newly proposed script of questions and ability genre levels approach to the profile module.

A3.1 Description

A user profile is required to determine the type of learning object that would best suit a learner, based on his/her learning style and abilities. An overview of this design and how it was implemented for version 1.0 is described below. When a user enters the DEAL project web site, the upper right hand side of the screen will have two links labeled "**`Login`**" and "**`Register`**" Fifure A3.1. Shows a screen shot of what the user will see.

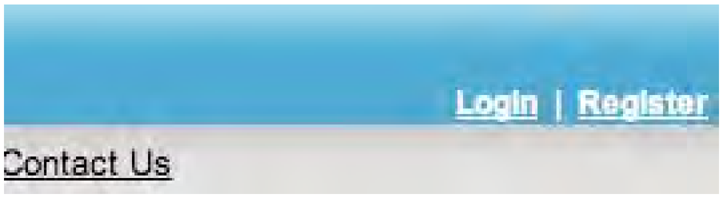
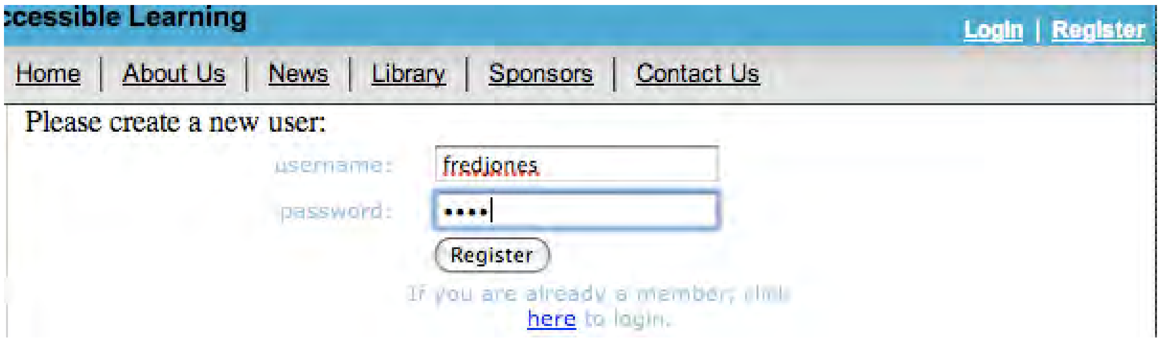


Figure A3-1: Links to Login or Register.

The profile module is currently designed to run automatically when a new user registers. It is auto loaded after a user selects a valid user name and password, and clicks the "Register" button As shown in Figure A3-2.



Please create a new user:

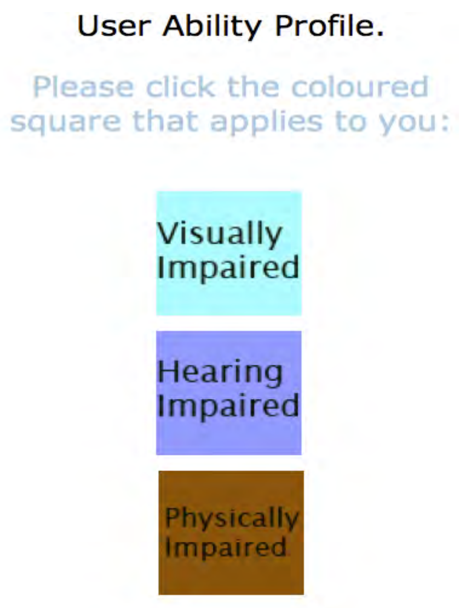
username: fredjones

password: ••••

If you are already a member, click here to login.

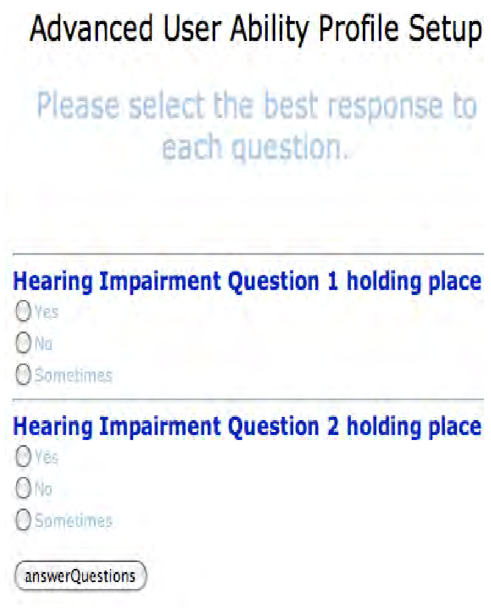
Figure A3-2: Register screen – click "Register" button to register.

For testing purpose: As it is shown in Figure A3-3, we have provided three types of abilities, and learner has to choose one of them. When the learner clicks the colored box that best applies to them, the system directs them to a secondary page of questions related to the severity of their capabilities as shown in Figure A3-4. This initial value is stored in the session object, and not in a database table since a learners profile is not actually complete until they successfully submit the next page.


User Ability Profile.

Please click the coloured square that applies to you:

Figure A3-3: Ability types


Advanced User Ability Profile Setup

Please select the best response to each question.

Hearing Impairment Question 1 holding place

◯ Yes
◯ No
◯ Sometimes

Hearing Impairment Question 2 holding place

◯ Yes
◯ No
◯ Sometimes

Figure A3-4: Final profile page

This secondary profile page shown in Figure A3-4 currently only contains temporary placeholders. (Creating the list of question is part of our future work). The information collected then gets submitted to the DEAL framework project. The storage of the profile data has not been implemented yet; however, the mechanism exists for such an implementation, and would be refernced with a command similar to the following:

`$this->engine->session->user->profile->get() and set()` to set user profile vars. The above code is currently commented out in the existing **`mod_userprofile2`** class.

The call to this session's user profile's getters and setters stores any user profile data in the "**`user_data`**" table of the database. This is a four column table, storing the **`uid`**, **`key`** and **`value`** to a unique **`userData`** id.

The proposed new profile, described later in this manual, will use this same "**`user_data`**" table, but will generate a single key value, representing this users unique ability profile instead of storing the results to every specific question.

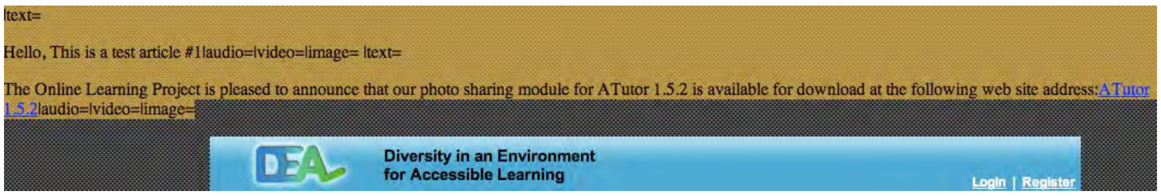
The metadata that results is then obtained by the DEAL framework engine and placed directly on the template that is being manipulated, on its way to the browser.

Current template metadata access

Currently, the values that the template displays representing the metadata are NOT in MMP-correct syntax. The MMP required profile metadata format was not known when this was initially designed, so the proof of concept does the following:

- Engine::**`getMetaData()`** This calls five MetaOption::getters. The final four return URLs of that learning object.
    - **`getOptionName()`** This currently refers to the news article.
    - **`getText()`** If profile indicates text to be appropriate, text obtained.
    - **`getAudio()`** If profile indicates audio to be appropriate, audio obtained.
    - **`getVideo()`** If profile indicates video to be appropriate, video obtained.
    - **`getImage()`** If profile indicates image to be appropriate, image obtained.
- A combination of any/all getter data above is possible, and all vars are echoed to template
- Engine::**`getFinalTemplate()`** Final call by Engine in a page load.
    - Places variables on temlate, at very top above global header
    - Displays template to browser

NOTE: When we implemented this, we choose the news topics as the default learning objects. In this case, as Figure A3-5 shows, every news story will have a variable entry at the top of the page.



```
|text=

Hello, This is a test article #1|audio=|video=|image= |text=

The Online Learning Project is pleased to announce that our photo sharing module for ATutor 1.5.2 is available for download at the following web site address:ATutor
1.5.2|audio=|video=|image=
```

Figure A3-5: A complete current profile template metadata dump.

The resulting template contains these metadata variables, visible to the user. When the intentional logic MMP is utilized, the goal is to have MMP use this metadata and a combination of the URLs that the metadata provides, in order to return the "**Best Fit**" learning object to the learner, via the template, based on their ability profile.

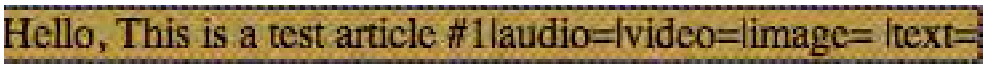

Hello, This is a test article #1|audio=|video=|image= |text=

Figure A3-6: A specific metadata profile

Teaching and Learning Styles

There are a plethora of models for learning styles. However, for the current research it is proposed to use the Dunn and Dunn's model which claims that visual learners have a preference for seeing (think in pictures; visual aids such as overhead slides, diagrams, handouts, etc.). Auditory learners best learn through listening (lectures, discussions, tapes, etc.). Tactile learners prefer to learn via experience—moving, touching, and doing (active exploration of the world; science projects; experiments, etc.). It is proposed that by designing web-based instructional content that accommodates each of these learning styles based on the individualized profile optimizes the web-based learning experience. This aspect addresses the diversity issues proposed in this research. The following table summarizes a study by Rajiv J. Kapadia [3] that was focused on engineering students. The case study attempted to understand the students learning styles, and then modify the delivery of information to match the student's learning styles. A high level summary of the different learning and teaching styles can be found in Table A3-1.

Learning Styles	**Teaching Styles**
Sensory	Concrete
Intuitive	Abstract

[3] Rajiv J. Kapadia , 2009,Teaching and Learning Styles in Engineering Education, 38th ASEE/IEEE Frontiers in Education Conference, October 22 – 25, 2008, Saratoga Springs, NY

Visual	Visual
Auditory	Verbal
Inductive	Inductive
Deductive	Deductive
Active	Active
Reflective	Passive
Sequential	Sequential
Global	Global

Table A3-1: Learning styles match up for Teaching styles.

Sensory – Love facts/figures. Slow methodical learners, taking longer to grasp concepts. They get the picture on lab completion, or final lecture. Love ‘turn the crank’ exercises.

Intuitive – Don’t like facts/figures. Love symbols and simulations. Often solve problems in unique way. This is a common learning type in engineering classes.

Visual – Don’t like repetition. Love diagrams, flow charts, demos, graphs. Students usually like to write things down. Typically take longest to finish tests. Refer to texts frequently. Most college age adults are at least partially visual.

Auditory – Love lectures and have great memories. Most classes are taught this way, which clashes with other learning styles specially the visual types. Auditory learners even prefer verbal instructions and guidance in lab settings.

Inductive – Most students think they are inductive learners. “Learning by example” and inferring a general rule from observed phenomenon are inductive approaches. Physics classes, and science more generally, use this approach with great success.

Deductive – Natural human learning and teaching style. They investigate and learn a lot on their own. They take concise notes and use only prescribed texts. They like facts/figures, but use to reach their own conclusion. Most school curriculums are laid out in this manner.

Active – Love some groups, and small problems to be done individually or within a team. Love 'one minute papers'.

Reflective – A methodical learning style. This type of learner focuses not only on learning, but the concept of learning. They like to understand 'why' they are learning it; 'how' they are learning it; and 'how' it can be applied; and, what strengths do they show etc.

Sequential – Broadly refers to the ordered blocks of lectures, quizzes, reviews, and tests that make up most curriculums. Sequential learners love this structured path, and course outlines are a sequential learner's best friend.

Global – Learn in spurts. Once they get it, it's retained and they even outpace other learning types at this stage. School can be difficult for these types. Periodic hard challenges with advanced concepts are great for global learners. A summary statement, and establishment of the larger picture in context helps a lot.

Kapadia concludes that a variety of simple teaching adaptations can actually cover almost the entire range of learning styles in some manner.

A3.2 Profile Module Plan

In contrast to the implemented beta version of the ability profile and templates displaying metadata representing the ability profiles, this proposed solution will NOT be passing any complex data such as URL's to the template and the intentionalized logic MMP layer.

Instead, it will use a more complex script of questions, and an analysis of the answers to these questions to create a simple variable. This simple variable will be what is displayed via the template, in correct MMP syntax, allowing the MMP logic layer to read the variables. In this proposed profile module, the variable passed to the MMP logic layer will allow that layer to determine what sort of learning objects to return, and where to retrieve them from.

This decreases the coupling of the framework from learning object storage location awareness, allowing it to concentrate on the analysis of the answers from the new ability profile question script and the creation of a single variable representing this ability profile. An example of a learning profile might look something like this. The following MMP macro called **`profile`** with values resulting from the profile questionnaire

```
{: profile}100.30.0{/:}
```

The results of the script are parsed and a single variable results. This single variable indicates a ratio of ability impairment from 1-100 for each of the initial ability genres “visual”, “auditory”, and “physical”.

This could represent a user’s ability profile that is visually impaired, 30% hearing loss, and with no physical impairment. 100 represent 100/100, which means fully visually impaired, and legally blind. 30 represent 30/100, which means less than moderately hearing impaired. 0 represents no physical impairment at 0/100.

Alternatively, each ability genre could easily have its own MMP tag, like this:

```
{: visual}100/100{/:}{: hearing}30/100{/:}{: physical}0/100{/:}
```

When appropriate, the MMP intentional logic will use the provided parameters to determine, what kind of learning object to return.

A3.3 Storage of metadata

If this profile module design is implemented, the modules “mod_userprofile” and “mod_userprofile2” will be combined to one, and will contain the question script. Submitting this script will cause the program to create the unique user profile variable data.

This variable can be stored using the existing method call through session’s user profile object.

`$this->engine->session->user->profile->get() and set()` to set user profile var.

Instead of storing the result of each answer like this, only the results of the analysis of the entire profile question script is stored.

This stored profile variable information is utilized every time a page object, is requested. For research implementation, a page object could be a news article as the beta profile module used, any html element such as a div block, or any portion of any learning object or combination of learning objects. The entire purpose of obtaining the profile metadata is to utilize it for custom content delivery, with inclusive design in mind.

NOTE: These rankings MUST be securely stored; This is sensitive data, being used for research and specialized content delivery of learning objects on a small scale.