Chapter XI
A Pliant-Based Software Tool for Courseware Development

Marcus Vinicius dos Santos
Ryerson University, Canada

Isaac Woungang
Ryerson University, Canada

Moses Nyongwa
University of Manitoba CUSB, Canada

ABSTRACT

The increasing importance of e-learning has been a boosting element for the emergence of Internet-based educational tools. As we move into the information age, tremendous efforts are made in the development of new information and communication technologies for educational purposes. The ultimate goal is to facilitate e-learning methodologies and acquisition. The chapter's contribution is in the area of open source software for technology-enhanced learning. First, we report on the capabilities of Pliant, a novel software framework for Web-based courseware development. Pliant' design features upon which e-learning capabilities are built are presented, showing that Pliant has some advantages over existing software, including flexibility, efficiency, and universal usability. A case study of the use of Pliant in the project "Multilanguage Database for Localization" developed at the CUSB School of Translation is presented. Second, we present Academia,[3] a Pliant-based courseware development Web portal, and its use in translation studies at CUSB.

INTRODUCTION

The widespread availability of Web-based educational systems and standard-based courseware systems, and their deployment in educational institutions, including educational community as a whole, has raised a clear concern regarding their "universal usability" scope (Hochheiser & Shneiderman, 2001). A thorough analysis of the situation and informal discussions with "on-line teachers" and teacher educators show that Web-based educational tools are quite far from achieving their main goal—that is, being used by a wide distance audience in a cost-effective and educationally sound manner, and in particular, endowing Web literacy to both young, old, novice, expert, and end users with less computing background. This chapter reports on the capabilities of Pliant, a high-level and flexible programming language and Web development framework. It shows how Pliant can be used both for high-level programming and e-learning purposes, while meeting educational and software-oriented expectations. Academia, an example of an open-source, lightweight, Web-based courseware tool fully implemented in Pliant, is presented. This portal has been designed to help instructors quickly create, post, manage, and deliver Web-based courses and other e-learning resources. A case study of the usability of Academia at a Canadian institution is presented. This Pliant-driven application is meant to show the efficiency of the Pliant's framework as a supporting tool for e-learning methodologies and acquisition.

The chapter is organized as follows. First, we briefly introduce the main streams driving the development of Web-based educational tools, and situate Pliant in that context. We then present an overview of the Pliant approach in terms of language constructs—here, we present our view of the Pliant architecture, and its underlying design features upon which e-learning capabilities can be supported. Next, we discuss various e-learning capabilities of Pliant, while highlighting their relationships to some of the main topics of this book. These include:

a. A description of Pliant as a tool for consolidating e-learning methodologies/acquisition—here, elements for exploration, data management, teaching, communications, and users' management are presented;
b. A description of Pliant as a tool for learning programming languages and Web programming—a case study of the use of Pliant in a project entitled "Multilanguage Database for Localization," developed at the CUSB School of Translation, is also presented; and
c. A description of Academia—here, our focus is on showing how this portal has been used as a tool for supporting translation studies at the CUSB School of Translation.

We also introduce Co-op Web,[4] a Pliant-based Web portal developed at Ryerson University, Canada, used to administer the Cooperative Education and Internship Program.

Some shortcomings of our framework and how these can be addressed as future research themes are then offered, in the perspective of enhancing e-learning methodologies and acquisition. Future developments of our framework are also highlighted, and finally, our conclusion synthesizes our discussion and presents our final remarks on Pliant's e-learning features.

BACKGROUND

Web-Based Educational Tools

The exponentially increasing number of educational courses being offered over the Web has spurred a growing industry of software tools to assist in the creation of Web-based curriculum and in performing class management tasks. For this reason, Web-based educational tools and standard courseware systems are two main research and development streams in the field of e-learning. The development of these systems can be categorized in two complementary streams. The first stream is based on the traditional approach of "hardwiring" high-quality educational material items in the course content—that is, the learning content used by the student resides in the system. Well-known examples of course management systems built upon this approach are Blackboard (2002) and WebCT (2002). The second stream is based on an adaptive approach, where a model of goals, preferences, and knowledge of each individual student is built and then used throughout the interaction with the student in order to adapt to the needs of that particular student. In this case, the learning content does not reside in the system, but in other distributed servers. Independently of the approach used, the majority of Web-based educational systems are based on technologies developed in the areas of hypermedia and intelligent tutoring systems (Brusilovsky, Stock, & Strapparava, 2000; Brusilovsky, 2001). The Pliant-based educational tool introduced in this chapter falls within the first stream. Pliant is a standalone and Web-based language, which encapsulates both the "human" and "computer" levels of thinking and coding programs. This unique privilege makes it an exceptional language, compared to any other existing one. It demonstrates that Pliant has a higher level of flexibility, adaptability, and integration, providing for higher software development capabilities and enhancements. Thanks to Pliant's HTTP (hypertext transfer protocol) server power, including server-side rendering, any mainstream Web browser should always be enough to access Pliant's documents, and by extension, Pliant e-learning materials. These features can be used to develop standalone Web portals for teaching/educational purposes, or to improve the current state-of-the-art e-learning development tools, while maintaining educational expectations, and economical, time constraints, and human resources limitations.

MAIN FOCUS OF THE CHAPTER

An Overview of the Pliant Approach

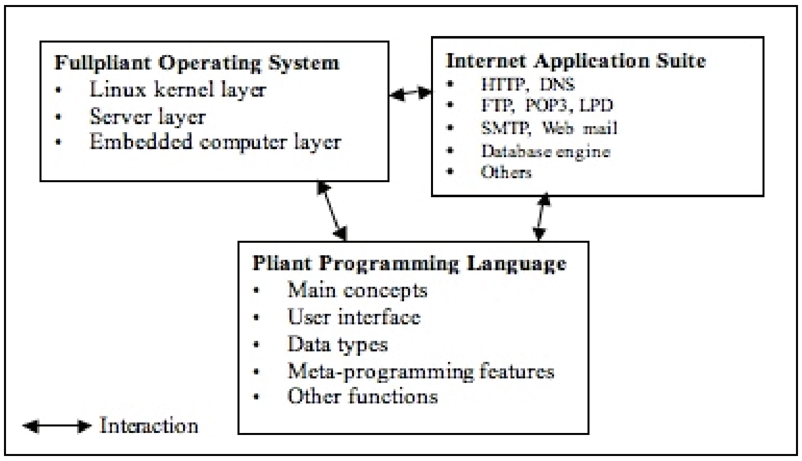
The Pliant project[5] was initiated in 1984 by Hubert Tonneau (Tonneau, De Mendez, & Santos, 1984). In his analysis of numerous software developments, Tonneau realized

1. The lack of coherence between applications, libraries, and so forth often required a large amount of glue between relevant pieces of code.
2. It seemed impossible to conciliate high-level constructions allowing improved expressiveness and conciseness in specific contexts, with low-level adaptability allowing efficiency and optimized handling of exceptional cases.

From these considerations, the introduction of a new, efficient, and multi-level language with a flexible syntax and structure, which could be adapted to particular application contexts, seemed appropriate. The Pliant language is thus oriented towards efficiency, understood in terms of computational resources, as well as programming adaptability (De Mendez, De Mendez, Santos, & Tonneau, 2000). The main design structures of the language can be described as modularity, dynamic compilation, and full reflexivity, allowing for the redefinition of the syntactical, compilation, and code optimization rules. New application services have then been integrated at the language level (good examples are scheduling primitives and database management), hence suppressing usual gaps and interfaces between applications. From this point of view, an application is seen as a set of libraries, or even as a language extension possibly introducing its own syntactical changes. These applications may also be gathered into a coherent execution context, leading to an actual operating system, called *Fullpliant* (whose source code is a size of 4.2 Mb only). This framework can be executed in two different ways: as a program executing various servers (on Linux or Windows platforms), or as an operating system lying on top of a Linux kernel. Pliant comes with many pre-built servers including DNS (Domain Name System), FTP (File Transfer Protocol), POP3 (Post Office Protocol version 3), HTTP, SMTP (Simple Mail Transfer Protocol), LPD (Line Printer Daemon protocol), remote execution, secured channel, and a database engine. The HTTP multi-site Web server provides the standard application interface. A powerful server-side dynamic page mechanism has been introduced, on which existing applications (Forum, photography correction and high-fidelity printing, Web-mail, etc.) rely, as well as additional HTTP-related servers (such as Web-based Distributed Authoring and Versioning–WebDAV). The limitation of the HTML (HyperText Markup Language)/Javascript scheme has led to the introduction of an enhanced extended Pliant browser, valuable as a state-of-the-art user interface for possibly distributed application software. Being used in an industrial context since 2000, *Fullpliant* is also concerned with security issues. The transparent integration in the dynamic page extension of signature and right verification mechanisms obviously demonstrates that security may be achieved without unnecessary additional programming complexity. Pliant (De Mendez et al., 2000) may thus be seen as a triad (see Figure 1): (1) the *Pliant programming language* and low-level libraries, (2) an *Internet applications suite*, and (3) the *Fullpliant operating system.*

The *Pliant programming language* is human oriented, that is, its syntax is trivial and strongly typed. Its expressiveness allows the user to program in a high abstraction level. The language is also reflexive, allowing the user to change the way Pliant parses and compiles expressions. In other words, users have a high degree of freedom to redefine the Pliant language itself, should they dislike a particular feature of the language or want to extend it (De Mendez, 1998). In addition to these meta-programming[6] capabilities of Pliant, the framework includes other modern programming principles, such as static typing, dynamic objects, lazy evaluation, reflective compiling, reference counting garbage collection, built-in debugger utility, scalability from low-level systems programming to high-level control languages, and an easy-following syntax. The Pliant compiler is dynamic and efficient, producing code, on-the-fly, as efficient as the best C compilers. The *Pliant Internet applications suite* consists of a set of

Figure 1. Pliant architecture



servers, a database engine, data encryption tools, and a handful of Web application tools, such as the HTTP server configuration tool, an online forum, Web mail, and printer configuration tool, to name a few. In the core of the suite is the Pliant HTTP server. It is in charge of hosting and dynamically translating Pliant Web pages, written in a subset of the Pliant language called the page format into HTML. The *Fullpliant operating system* (to be used by advanced users) has two main goals: facilitate the system administration of a set of computers, and make it easy to automate repetitive tasks. It appears from the above description that the two key design features of Pliant, upon which e-learning capabilities can be developed and supported, are the Pliant' intrinsic meta-programming constructs and the Pliant' ability to accommodate several multi-site Web servers.

Pliant' E-Learning Capabilities

E-learning can be defined as the use of network technology to design, deliver, select, administer, and adapt learning activities. Here, two basic types of technological solutions can be used: synchronous model (such as audio-video streaming and videoconference), and asynchronous model (such as hypertext publication). Existing e-learning management systems such as WebCT and Blackboard incorporate both models and corresponding services in different ways. However, due to certain limitations inherent to system stability, troubleshooting, cost-reduction, file format accommodations, Web browsers, customization, and others, none of these platforms includes a support to help manage the dynamics of e-learning activities. In an attempt to overcome these disadvantages, studies of management issues in e-learning environments has become critical for the success of Internet-based educational services. A relatively recent work on the management of e-learning environments has shown the effectiveness of using the workflow concepts (E-Workflow, 2003), techniques, and tools to help manage e-learning. Under this concept, the e-learning process is considered as a two-level interconnected process: the e-learning environment and the e-learning activities. The e-learning environment is further broken down into five different phases, each with its own set of tasks. These are:

1. **The conceptual phase:** Course subject, target audience, contents, budget organization, and so forth;
2. **The planning phase:** Details for the establishment and preparation of a specific instance of an e-learning action;
3. **The execution phase:** Period of time during which the students are active in the learning process; and
4. **The procedural evaluation phase:** An analysis of how the e-learning fulfilled its aims; additionally, the workflow e-learning environment model incorporates features such as improved efficiency, better process control (i.e., standardization of working methods), improved users service (i.e., greater predictability in levels of response to users), flexibility (i.e., ease of redesigning in line with changing needs); and
5. **The business process improvement:** Streamlining and simplification of processes.

The e-learning activities are concerned with the monitoring of actions and interactions among the above described phases. They are controlled by means of *learning objects.* The efficiency of an e-learning management system using workflow is measured by its capability in reusing learning objects. In this respect, workflow software and XML (Extensible Markup Language) are viable tools for describing learning objects. Any attempt to provide implementation techniques that could result in promoting the deployment of reusable contents or learning objects for e-learning purpose, while enhancing the integration capability of its underlying software platform, is therefore highly desirable for both the providers of educational services and the e-learning research community. Pliant seems to be a suitable candidate able to fulfill these requirements because of the versatility of the language constructs.

Pliant as a Tool for Consolidating E-Learning Management Systems Platforms

Any e-learning management system is driven by its underlying software platform (i.e., the set of programs that provide functionality to the application). At a human level, programs appear as algorithms, that is, a list of tasks, each being expressed by a single word or by a full sentence with subordinate clauses referring to subtasks. At a high (symbolic) level, a program is stated as a list of expressions. Each expression is a piece of raw data characterized by its semantics. At a low (i.e., code) level, a program is a set of instructions, where each instruction is a function call with a set of parameters. At this level, no ambiguity should remain.

A programming language is a bridge between the human way of thinking of an algorithm and the computer way of coding a program. Prior languages, including those on which the existing e-learning management systems are built, focus either on the semantics, but fail to be efficient at code level, or on efficiency, hence failing to be intuitive and easy to use by a human. To our knowledge, Pliant is the *first language* that makes the "single language" option a possible one by acting as a bridge between the aforementioned two levels. We assert that Pliant is the best one available, because it addresses this bridging goal at the highest level of flexibility and the best level of efficiency. This ability to write all code using a "single language" means better internal communication, time-saving improvements, load sharing, and shorter code. It also means reasonable scalability, adaptive user interface, easy switching from a closed software-like model to an open software-like model, more flexibility, customization, development power, strong design and high-quality program, low cost, dynamic highly reflexive compiler, less hardware limitations, adaptive hardware, and so forth. The list of goods is long. In order words, Pliant tries to bring as much expression power as possible, without impacting low-level code performance. These capabilities allow Pliant to be seen as a kit that greatly simplifies the distribution of software. In addition, Pliant can also offer e-learning potential, such as Internet-based learning and didactical requirements.

As a developed Web technology, Pliant is an Internet suite containing material needed to start an Internet site, including a database engine, a forum, a graphical toolkit, dynamic pages, and mail support. Therefore, it provides a suitable multimedia support to teachers and students, just as other proven e-learning systems, but with the added flexibility and adaptability of the underlying software and hardware platforms as pointed out previously. Pliant provides the choice to select elements according to the needs of teachers and students, and independently of their program of study. These elements can be divided into three interrelated groups, classified as: (1) elements for exploration and data management, (2) elements for teaching and communications, (3) and elements for users' management.

Elements for Exploration and Data Management

Pliant allows for a selection of various types of advanced Web browsers—Netscape, Mozilla, Internet Explorer, Konqueror—and is open to other better ones available. The choice of a browser depends on the security, portability, and computer power requirements. It also provides various types of servers suitable for Internetworking, such as HTTP, FTP, DNS, SMTP, POP3, Web mail, backup system, files browser, database engine, and so forth.

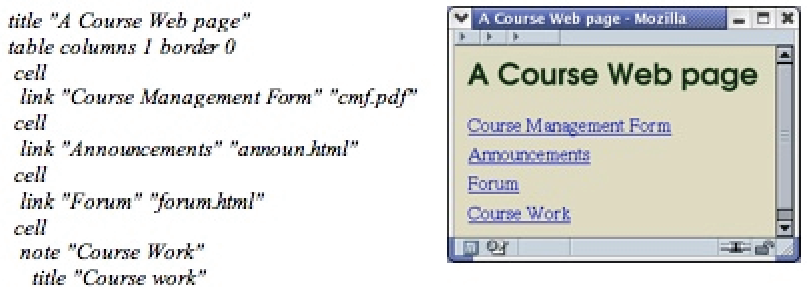
Elements for Teaching and Communications

Pliant can be used as a tool for enhancing course management systems. It provides a flexible software support for a variety of learning processes such as the distribution of documents and communications through the Internet. Pliant's online forum application provides asynchronous communication between instructors and students. The simple structure of the language allows one to easily create course Web sites on which one can post course notes for anytime access by students. With some training in Pliant programming, novices can quickly move on to the development of their own handy, tailor-made teaching tools, such as course assessments, online tests, and an online grade book. To illustrate the above points, we consider a simple case of a typical course Web page written in Pliant, and its results as shown in Figure 2.

Several interface options are provided:

1. *Access to course documents,* in this case exemplified by a link to the *course management form* and the course work, but can be a list of documents, such as readings, lecture notes, schedules, syllabi, course management forms, organization of course projects, priorities, and details, and so forth. It also supports the import and export capabilities by means of inserted Web site links, allowing the instructors to gain access to a complete set of teaching tools provided by academic publishers, or to create a package of the course content that can later be imported into another course.
2. *Course announcements,* a key place to put daily, weekly, or monthly time-sensitive course information such as deadline changes, clarifications, remainder of upcoming class chats, schedules, important events and dates, and so forth.
3. *Forum* is a Web portal that behaves as a virtual classroom and lightweight chat. It enables users to participate in an online collaboration with students and instructors. As a discussion board, messages are posted to the board, and every permitted user is able to read the messages and reply to them. Like a bulletin board, one copy of the message exists, and only the course designer has the right to delete the messages. A forum is a tool that fosters communication and collaboration as a way to enhance course material. Several forums can be created simultaneously, providing for a frame for team working. Each forum is assigned a thread (i.e., a discussion session) so that all replies to a given message are contained within the same thread. Within a forum, a messaging program is implemented that allows one to

Figure 2. Typical course Web page in Pliant



```
title "A Course Web page"
table columns 1 border 0
 cell
  link "Course Management Form" "cmf.pdf"
 cell
  link "Announcements" "announ.html"
 cell
  link "Forum" "forum.html"
 cell
  note "Course Work"
    title "Course work"
```

A Course Web page

Course Management Form
Announcements
Forum
Course Work

send e-mail messages to the users who are members of the forum, and to keep track of those messages. As a collaboration tool, a forum allows its users to enter into a real-time discussion with instructors, students, and colleagues; to access the Web; and to engage in question-and-answer sessions. The option of considering grouping student lists into several small groups can also be applied to keep the conversation manageable due to the synchronous nature of the discussion forum panel. It is important to point out that collaboration sessions throughout forums are recorded by means of subjects and messages. The leader of the session (course designer) must start the recorder to create an archive.

Elements for Users Management

In an e-learning prospective, the user management capabilities of Pliant is mostly reflected on the ways that Pliant enables the instructor to manage the users in their course sites. This involves the following types of setting privileges: enrolling users in the course, which means that the user must already have an account; removing users from the course; and creating groups of users within a course with the right to modify groups. The instructor has the option of giving the group access to its own private discussion board, virtual classroom, group file exchange, or group e-mail.

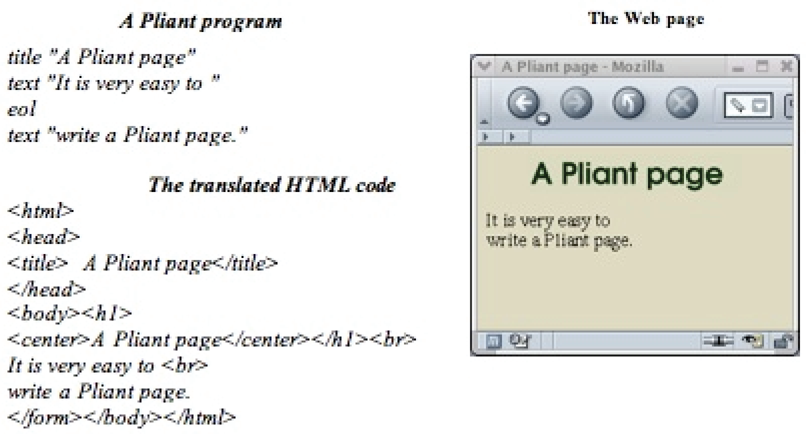
Pliant as a Tool for Learning Programming Languages and Web Programming

Due to the Pliant's meta-programming features, Pliant can be customized for learning and teaching purposes. Standard Pliant applications are browser based—that is, the programmer can host his or her Pliant program in the Pliant HTTP server and then interact with it using a Web browser. Browser-based Pliant programs have file extension *.page* and are written using Pliant's Web page programming instructions, called the *.page format,* an alternative to HTML/XML. Throughout the aforementioned interaction, the HTTP server works as follows: it keeps listening to requests from clients (i.e., other browsers on the Web); once it gets a request for a page, it translates the respective *.page* Pliant program into HTML and JavaScript code and sends it to the client (browser). If the client requests a plain HTML page hosted in the server, then the server simply sends it as it is. Hence, for the client, there is no difference; it is plain HTML/JavaScript coming from the server side. If the requested page does not point to a *.page* or static HTML file, then the Pliant HTTP server recursively searches for a file called *virtual_tree.page* in the path of the requested URL. This Pliant concept, called *virtual tree mechanism,* provides an easy mapping of data sets to URLs. Without it, HTTP options would be used—a not as clever and convenient solution. We relied heavily on such contrivance to implement one of Academia's[7] subsystems—the course Web content renderer.

To illustrate this simplicity and power of the Pliant language combined with the HTTP server, we now present short examples of Web applications written in Pliant. The Pliant program presented in Figure 3 illustrates how easy it is to write a simple static Web page with a title and some text. It shows a simplified HTML code generated by the Pliant HTTP server. The client will see this application as shown in Figure 3 (right side). As illustrated, Pliant has the capability of generating and caching online graphics when server-side font rendering has been requested.

The command title "A Pliant page" produces a page title; the command text, whose argument is a string, outputs text. Pliant provides a plethora of commands for writing Web pages. The interested reader should check the Pliant Documentation Initiative site (De Mendez et al., 2000). The next example illustrates how Pliant can

Figure 3. A Pliant program and its resulting Web page



A Pliant program

```
title "A Pliant page"
text "It is very easy to "
eol
text "write a Pliant page."
```

The translated HTML code

```
<html>
<head>
<title>   A Pliant page</title>
</head>
<body><h1>
<center>A Pliant page</center></h1><br>
It is very easy to <br>
write a Pliant page.
</form></body></html>
```

The Web page

```
A Pliant page - Mozilla

A Pliant page

It is very easy to
write a Pliant page.
```

be used to generate dynamic Web pages, such as a Web page for converting currency from Euros to Canadian Dollars. The programmer writes a page as follows:

```
title “Euro to Dollar”
  var Float euro := 1
input “Amount: “ euro
  button “Press me”
title “The answer is…
text (string euro*1.2)
```

The user types in the amount in Euros he or she wants to convert (left of Figure 4). When the user clicks on the button, a new page, whose code is defined in the shadow of the button (i.e., indented with respect to the button instruction), will present the result of converting the input value from Euros to dollars (right of Figure 4). Notice that the function call (*string euro**1.2) transforms the numerical result of the expression *euro**1.2 into a string.

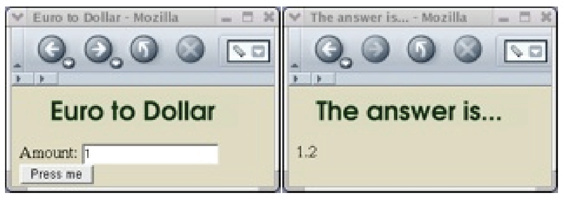
For programming languages development purposes, Pliant's default syntax is lighter than others because of the following main features:

1. Many parentheses are implied by indentation,
2. The ‘:’ operator replaces some extra parentheses, and
3. There is no ‘,’ operator to separate the parameters of a function.

These features, combined with the meta-programming ones, make Pliant a language of choice for teaching the programming concepts. Because the purpose of this chapter is not on “experiencing programming languages,” we will not elaborate much on Pliant's language specifications referred to the main concepts, user interface, data types, meta-programming optimizers, and other programming features. Interested readers can find detail information at http://old.fullpliant.org/pliant/language/.

In short, beginners can use Pliant as an interpreted language by writing small pieces of code

Figure 4. Dynamic Web page in Pliant



```
Euro to Dollar - Mozilla

Euro to Dollar

Amount: 1
[Press me]
```

```
The answer is... - Mozilla

The answer is...

1.2
```

and running them directly. Experienced programmers can run Pliant as a compiled language—that is, by writing efficient programs while using most high-level programming features of object-oriented languages and the expression power of logical programming languages. Pliant is also an ideal linker, in the sense that one can write different pieces of a project in different programming styles with all parts interacting.

Case Study on the Use of Pliant in the "Multilanguage Database for Localization" Project

In today's world of global operations and international technical communication, the Internet is fast becoming the primary port of call for information, education, training, and services. Consequently, more and more development initiatives go beyond local borders. Experienced professionals understand that to be effective, training must be done in the right language and with culturally appropriate resources and methodologies. In response, Canadian businesses, corporations, and universities have quickly become aware of the benefits of *localization* as one of the boosted agents shaping the new economy.

Localization can be defined as the process of taking a product and making it linguistically and culturally appropriate to a target locale (country, region, and language) where it is used and sold. This multi-layered process requires programming, linguistic skills, translation skills, cultural knowledge, and most importantly, an *e-learning development platform support.* It has been recognized as an important part of the educational program in Translation and Computer Science schools at Canadian universities and abroad, where students obtain basic education and training (both onsite and Web based) in both computational methods and localization techniques.

The increasing importance of terminology banks and translation memories in the translation process, a sub-component of the localization process, has created a need for developing language-based repositories for the purpose of using them in the localization training and practices. As a response to this deficiency, the "Multilanguage Database for Localization" project (Nyongwa & Aubin, 2004) was launched at the CUSB School of Translation, in collaboration with Ryerson University. The objective of this research is to develop a comprehensive database framework of languages which can efficiently support the localization training and practices. One subcomponent of this project entitled "Enhancement of the Pliant Language Database Engine" has been to investigate how the Pliant system, although not originally meant for language acquisition, can be used as a benchmark tool to support the user interface design of the aforementioned database framework, at least for the online localization training portion. In this context, thanks to its meta-programming feature, the Pliant language design has been altered and successfully tested to allow for customized content-building instructions. These later features were then used through *Academia—a Pliant-based courseware tool*,[8] to support the localization training in various capacities. Two scenarios are described later in this chapter to illustrate some of these capacities. The first scenario, described in the subsection entitled "Case Study of a Course Delivery," discusses a particular instance in the teaching of the "translation process" portion of the course in localization. The second scenario, described in the subsection entitled "Pliant System, Translation and CALL," illustrates how the "terminology" component of the localization course is taught using Academia.

Usability of Pliant in Teaching and Project Management Contexts

To further assess the usability of Pliant in teaching and project management contexts, we have implemented two Pliant-based applications. The first one is *Academia,* a courseware development

Web portal, and the second one is *Co-op Web,* a management Web portal. In the sequel, we describe the first application in-depth, followed by a brief description of the second one. The reason is that the second application does not directly serve the purpose of this book.

Why Academia?

For instructors to carry on the mundane activity of setting course Web pages, two options are possible: design new course Web sites from scratch or resort to a courseware tool. However, such a tool is usually expensive and complex. The source code is usually proprietary, that is, it does not allow end users to further customize the code to suit their needs. Academia can address these deficiencies. The reader may then ask why one would use Academia and not another well-known courseware tool that provides every single feature an instructor can possibly fancy. Our answer: because as a Pliant-based application, end users can build on Academia's design and source code to develop more advanced and customized courseware materials, since there is no need to resort to different languages (as traditional approaches do) to develop applications that involve dynamic pages and databases.

Academia Features

Courseware products such as Blackboard (2002) and WebCT (2002) will take HTML documents, along with other media and resources, and quickly organize them into a framework specifically designed for delivery of Web-based courses and other learning resources. Frequently, they are used to complement traditional lecture-based programs. Courseware products are helpful to educators who are unfamiliar with programming, allowing easy integration of password protection, interactive activities, tracking of student progress, and so forth. Overall, the interface is fairly simple for the designer, as many use templates and wizards extensively to assist in course content creation. Step-by-step guides support creation of a range of components, from the course homepage, to bulletin boards, to quizzes and marking systems. When compared to the aforementioned courseware tools, Academia is a much 'lighter' application in the sense that it provides only the essential features required to set up a simple Web page for a course. Such a Web page may include links to a locally hosted copy of the course syllabus, assignment descriptions, instructor's lecture notes, and other materials.

Academia's User Interface Design

The design concept behind Academia's user interface was: *it should be as simple as possible, but not simpler.* In the first stage of the design process, a list of the functionality required of the system to accomplish the goals of the project and the potential needs of the users was prepared. From the instructor, manager, and students' viewpoints, this list included the major functionalities illustrated in the user-case diagrams presented in Figure 5. In the second stage of the design process, an analysis of the potential users of the system was carried out through discussion with instructors who already had previous experience with other computer-based teaching tools, with instructors who had no previous experience with such systems, and with university students attending courses taught by the authors of this chapter. Typical questions presented to these individuals were of the sort:

- What would the user want the system to do?
- How would the system fit in with the user's normal workflow or daily activities?
- How technically savvy is the user and what similar systems does the user already use?
- What interface look-and-feel styles appeal to the user?

The answers were then compiled and cross-checked with the functionality list obtained in the respective analysis, resulting in a new minimal functionality set and overall look-and-feel style.

In the next stage, a site flow of the system that showed the hierarchy of the pages was developed. Next, prototyping and usability tests (Nielsen, 1993) were performed. In this stage, a proof-of-concept Web application was developed that showed the basic functionality set and content. Fast prototyping of the system was facilitated by the intrinsic features of the Pliant programming language. In regards to usability tests, the so-called *talk aloud protocol* (Wang, 2000), where you ask the user to talk about their thoughts during the experience, was performed via personal interviews with a small set of users—only instructors who volunteered to serve as guinea pigs during this tests—or 'looking over their shoulder' while they attempted to perform specific tasks with the system.

The ultimate goal of the Pliant software is "universal usability." As such, the Pliant user interface, or equivalently, the communication channel between the user and the functional elements of the system, has been intentionally made *text mode* based. This allows the user to focus on the task at hand and reduce the amount of overhead knowledge required to communicate effectively with Pliant and its underlying e-learning system (Academia). The function of the Pliant interface subsystem resumes in assigning user input to internal representations of Pliant's application and internal representations of the application to output that is comprehensible to the user. Thanks to the meta-programming features of the Pliant language and its reflective architecture, we believe that Pliant is compliant with the CALL aspects (i.e., the teaching and learning processes) with respect to online learning.

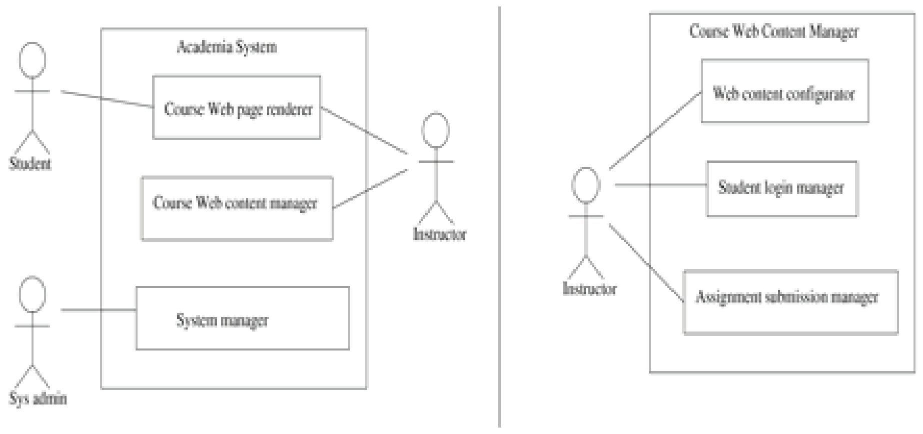
Academia's System-Level Capabilities

These are presented here by means of use cases and sequence diagrams. Such notations are commonly used in the area software engineering for describing a system without revealing or implying any particular implementation of the system (Booch, Rumnaugh, & Jacobson, 2005). The use case (left of Figure 5) presents Academia's subsystems and its actors.[9] The system administrator actor uses the *system manager subsystem* to keep track of the instructors registered in the system and the courses they teach. Instructors use the *course Web content subsystem* to set up the Web page of their courses. Once a course Web page has been configured, students can then visit the respective Web page, which is dynamically created by the *course Web page renderer.*

Course Web Content Manager Subsystem

Figure 5 (right side) depicts the course Web content manager subsystems. The *student login manager* subsystem allows the instructor to upload and manage the student's login access to the system. The *assignment submission manager* subsystem allows the instructor to view currently online submitted assignments and to download a compressed file containing all the assignment submissions. The *course Web content configurator* subsystem allows the instructor to interactively configure a template design for the course Web page. The instructor can define his or her contact information, such as office number, office hours, lecture times, and locations; upload course syllabi, assignment descriptions, and lecture notes to the server; add an announcement board to the course Web page; and set up an online discussion board for the course.

Figure 5. Academia system-level architecture



Course Web Page Renderer

The course Web page configurator subsystem stores the configuration settings of a course Web page template design in a database. This database indexes courses by their term, year, code, and section number. The course Web page renderer uses this information and Pliant's virtual tree mechanism (explained earlier) to retrieve and display the Web content stored for a particular course. More specifically, suppose that an instructor whose username is *john.doe* has already configured the Web content for the following course:

***Term**: Winter*
***Year**: 2005*
***Code**: Phil 660*
***Section**: 001*
***Name**: Philosophy of Love and Sex*

If a client (browser) requests the URL[10] *http://academia.org/browse/mycourses/john.doe/,* then the Academia course Web page renderer would dynamically create a Web page that lists links to all currently hosted courses of instructor *john.doe.* It does that by using the virtual tree mechanism because there is no *index.html* file in the above location on the server. The Pliant program that implements the course Web page renderer resides in the *virtual_tree.page* file, located at the root of the path /browse/. Therefore, when the client requests the above URL and the server finds the *virtual_tree.page* file in /browse/, it stores in the internal variable *virtual_path* the remainder subpath */mycourses/john.doe/* and runs the *virtual_tree.page* program. For this particular case in which the above path starts with *mycourses*, the renderer parses the path and extracts the relevant information needed for determining which courses to list on the dynamically created Web page.

A similar process takes place when a client (browser) requests the following URL: *http://academia.org/browse/courses/Winter/2005/Phil660/001/index.html.* However, no *index.html* (or *index.page*) file actually exists in the above location. Hence, analogous to what we explained above, the Pliant virtual path mechanism will again run the renderer. This time, the internal variable *virtual_path* will hold the sub-path */courses/Winter/2005/Phil660/001/.* Notice that this path provides all the information to locate the Web content for a course in Academia's database. The renderer parses this path to extract the course information (code, term, etc.). It then retrieves the Web content for the course from the database and dynamically creates a Web page for the course.

Towards Integrating Academia with Other Learning Management Systems

The initial idea underlying Academia's design was to provide an "*as simple as possible,*" but not simpler courseware development and e-learning tool. As such, integration with more robust learning management systems (LMSs) was not a priority during the development. Notwithstanding, all database files used or produced by Academia are in ASCII text, and information is stored in XML-like data structures. As an example, below we list an excerpt of a user database:

```
<pdata path="/user/marcus/contact">7062</
pdata>
<pdata path="/user/marcus/email">marcus.san-
tos@mac.com</pdata>
<pdata path="/user/marcus/first_
name">Marcus</pdata>
<pdata path="/user/marcus/homepage">www.
cs.ryerson.ca/m3santos</pdata>
<pdata path="/user/marcus/language">English</
pdata>
```

Notice that each entry in the database is encoded as simple XML code. Therefore, interfacing such data with an LMS would be straightforward. Pliant also includes many modern programming principles such as meta-programming, static typing, objects, reflective compiling, reference counting garbage collection, built-in debugger, clean syntax, and many pre-built components such as HTTP, FTP, SMTP, POP3 servers, a tree-based distributed data model, and a database engine. The intrinsic features of these core components can be used to support open Internet standards such as ECMAScript, thus, a-fortiori, to facilitate the compliance of Pliant with SCORM, but there still a long way to go to achieve this goal, and we have left it for future work.

A Case Study of Academia

This section reports on the findings from a study to experimentally compare some available Web-based learning tools used for the CUSB Certificate Program in Translation. These findings are discussed in relation to basic usability issues of Web-based tools, in terms of online course pedagogy, technological infrastructure, and students' perceptions. Our investigation attempts to justify the use of Academia as an e-learning methodology and acquisition tool.

The Context

Translation is a professional activity that requires both mental and physical settings, where profits and ethics must be met, just like in any other profession. Becoming a professional translator requires hard work, curiosity, open-mindedness, and experience. Based on these basic facts, the School of Translation at CUSB foresaw the importance of Web-based education in the mid-1990s, then introduced a few online courses in its Translation program. By the year 2000, the entire Certificate Program, composed of 10 three-credit courses, was successfully launched. This program (referred to as the *TOP* program) was designed to accommodate the draft curricula for various areas of expertise within translation and the actual needs of the translation industry. It has been set up primarily for people who are interested in studying translation while maintaining their current employment. It was particularly aimed at those who work in remote areas and would find it impossible to attend classes at a local university. At each session, students may decide to take only one course at a time, or more, depending on their own personal timetable. A number of Internet-based courses, including *Localization,* were gradually introduced into the TOP program.

Technological Infrastructure

The CUSB infrastructure is made up of three servers, a dozen PCs for instructors, and a 10-PC laboratory for students. There are also three other high-tech laboratories and a multimedia center where on-campus students can work. Several programming tools and Blackboard (2002) are used by students and professors to develop and manage Web-related documents. Although this courseware tool was greatly appreciated by the students and staff, complaints were quickly raised, mostly on usability issues—such as long login sequence and difficult navigation. To address these drawbacks, Academia was used to supplement Blackboard.

Online Course Methodology

Unlike traditional courses, online courses are space and time independent. Meanwhile, in the case of the TOP program courses, a framework has been developed that meets the traditional division of an academic year—into sessions of 15 weeks each. Courses are delivered according to this division. The content of each course is organized as followed: knowledge review, lectures, practices, debates and discussions in the forum, exchanges of e-mails with instructors, and evaluation. Courses content is built into modules to facilitate individual learning. The evaluation consists of four formal tests, one in the fourth week, the second in the eighth week, the third in the twelfth week, and the last one in the fifteenth week.

Case Study of a Course Delivery

To demonstrate the capability of Academia as a support for e-learning methodology and acquisition, we have implemented an instance of a particular course entitled "Localization" on a specific problem: the problem of encoding/decoding the characters in a text file during the lifetime of the localization process. Localization is a process by which a product (in this case a text file) should be adequately adapted to the characteristics of each country and equipped with a presentation that is acceptable at least at the level already reached for its original locale. This process requires various types of operations involving the targeted language and written communication (localization of translations), as well as the supported technical infrastructure for data processing (software localization). Both types of localization sub-processes are technology dependent and are mandatory in a training program in Translation such as the TOP program. Here, we focus only on the software localization process. One of the major problems when running this process is the lack of a clear mechanism that ensures the identification of the encoding used to save/open any type of document, or to escape extended characters that are not supported by the targeted encoding technique. This issue is particularly important in this context since it determines translatable and non-translatable data. Among proposed solutions, XML has been proposed as a viable one (Savourel, 2000) for the implementation of a multilingual solution. However, handling XML-like data structure files within the localization process is still a difficult task. To circumvent this difficulty, the Pliant-based capabilities of Academia are embodied in new standards or software processes such as XPath (Extensible Path), which provide several new features of dealing with data in general and localizable text in particular, based on the file format. Many state-of-the-art translator tools, available at *http://www.w3.org/,* are then used to apply the above mechanism in a concrete example. Depending on the power of the translator tool, the *Fullpliant,* and the type of XML file to be translated, the required steps are:

1. *Mark up the information in the text to be localized,* that is, find answers to the following questions: What text is translatable? Which language is chosen? Which local is referred to? What are the acronyms explanations?

What constitutes the verbals and nominals in short sentences of the targeted text?

2. Using the Translator tool, *create an XSL template (document) with appropriate parameters.* This is achieved by writing a skeleton of HTML elements and by using the XSL style sheet commands to provide the text.
3. *Process the file to be localized by using the above template and by choosing the locale to work with.* Steps 1 and 2 are not always straightforward and need some good understanding of XML and XPath.

The following methodology was used. Prior to the localization course, students are given some basics in HTML and XML programming. They are introduced to the structural power of both languages and to Academia as a programming platform. Their attention is then focused on the simplicity of XML, and the way its features can be used to extract the translatable and non-translatable text in the file format to be localized. Students are taught, by means of examples, on how XML code can be written based on the file format and content, and then structured in a way that the translator can use it to generate the desired output. Students are then asked to compare the XML and corresponding HTML files. Then, the localization course is taught to students through different modules (process, Web site, project management, etc.), providing them with necessary material to tackle other steps of the translation process. At the end, students are asked to extract HTML and XML files in English and to localize them in Canadian French.

Findings

The separation of the content from the format allows the translation task to move faster while reducing the time allocated to pre-translation and post-translation processing. Due to the expressive power capabilities of the Pliant-based framework upon which Academia is built, students have gained a great level of confidence when integrating the localization information as a component of XML, rendering the translation process more achievable than ever. This integration would have been more difficult without the use of built-in and enticing capabilities for data processing provided by the Academia platform, which greatly simplifies the entire localization process.

Pliant System, Translation, and CALL

Even though the Pliant system was not originally meant for language acquisition, the flexibility it provides in terms of integration features makes it possible. In technical translation and specialty languages teaching, terminology is an essential topic that should be studied. The teaching of terminology in the context of the localization course using Academia has been addressed in a CALL-like methodology manner. For a *given text* that needs to be translated from one language to another, the *steps* followed by the students, as well as the corresponding *Pliant-based methods* implemented to achieve these steps, are shown below.

1. Identify and establish a list of "appropriate" terms from the text to be translated. This task is achieved through the design of the Data Discovery Module (DDM). Here, some predefined language-based metrics are used to identify the aforementioned terms, and a Pliant-based user-interface is developed to extract these terms and stored them in DDM.
2. Ask the students to search for two or three contexts of the usage of each term from an established list obtained from various terminology repositories. This task is achieved through the design of the Data Context Module (DCM), where dictionaries and textual databases are stored. It mandates

the implementation of a Pliant-based Web interface for data search and retrieval (using the Pliant mainstream servers) in these repositories.

3. Search for equivalent terms in the target language through available dictionaries and databases. This task is achieved by means of the Pliant-based Web interface for data search and retrieval described in Step 2.
4. Divide the text to be translated into individual sentences to be translated, and then allocate one sentence per student. This task is achieved through the design of the Division Module (DM). A Pliant-based script is developed to build this module.
5. Each sentence is translated by a student and sent to the discussion forum for revision by other students within the group. This task is achieved through the design of the Translation Module (TM). A Pliant-based forum is designed and implemented to handle the communication and transfer of information between all participants.
6. The revised sentences are put together again to create the translated text. This task is achieved through the design of the Assembly Module (AM). A Pliant-based script is developed to build this module.
7. The translated text is sent to the instructor for evaluation, verification, and validation. This task is achieved through the design of the Quality Assessment Module (QAM)—a set of Pliant-based evaluation and validation methods (or functions). This module also provides the potential for integrating the Pliant system in a distributed environment.

Co-Op Web

The Department of Computer Science at Ryerson University offers a five-year bachelor's program in Computer Science with a Co-op option (CSCC). The program requires the management of students' applications, admissions, career planning resources, course selection requirements, graduation requirements, financial requirements, and job placements, to name a few. To efficiently address these challenges, Co-op Web was developed using Pliant; it is currently use to administer the CSCC program and has shown great satisfaction from its usability' point of view.

Actual Management Aspects of Pliant after Development

Our current challenge remains to understand the experiences of instructors (and students) as they adopt Academia as a course management system and integrate it into their teaching (respectively learning), either in isolation or as a complement to existing learning management systems. Our plan is to study several patterns explaining how instructors/students experiment with individual features of Academia, facing both technical and integration challenges, and attempting to adapt Blackboard or any other sophisticated learning system to match their goals and practices. Academia has become "mission critical" in fulfilling some of the teaching and learning central goals: enriching the student/instructor learning experience and advancing access to resources for teaching, learning, and research. Academia's training programs are currently being handled throughout the university, in parallel with Blackboard training programs, where faculty and students can receive assistance with all aspects of Academia and Blackboard operations. Several upgrades have been done and are continuously done to the Pliant framework in order to solidify its underlying e-learning capabilities and integration features, both transparent through Academia. For example, at the departmental site, faculty members can now upload their final grades from a Blackboard grade book into a Web grade roster, instead of having to enter them manually. A tracker has also been set to report general bugs, feature requests, fixes, and other issues such as sensitive security problems.

In its current form, the dynamic compilation structure of Pliant offers many opportunities of experimentation for academic and industrial research as well. Thanks to Pliant's control over generated codes, academic and industrial fields used Pliant for physical modeling purposes. For example, in thermic equilibrium when injecting thermoplastic material in a mould, Pliant was used to update some models without having to recompile the system. In Mathematics, Pliant was used to re-implement some graph manipulation tools.

The integration of Pliant with SCORM, NLN, and IMS format learning objects is possible due to the dynamic compilation architecture of Pliant and its reflectivity. This option is currently under investigation. Another current application scenario is the use of Pliant to manage distributed systems. This has been achieved so far in some cases thanks to the meta-programming feature of Pliant and its tree-based distributed data model, which allows one to maintain a database of hardware available on each machine and of software to be deployed there.

To circumvent some of the drawbacks of Pliant's architecture, it is suggested that we look at some high-level optimization algorithms in academic literature and implement them in the Pliant language, the goal being to rewrite the entire framework in Pliant. This will facilitate the pliant deployment, its integration with other systems including learning management systems, and its ability to embrace and extend existing services, as well as enhancing Pliant's robustness in terms of code generation and reusability.

FUTURE TRENDS

The Pliant *.page* mechanism, which we have used to implement Academia, has proven very convenient for quickly prototyping and writing simple user interfaces. However, the current (and future) trend is to provide programming languages and Web application development platforms rich in interactive features, such as graphical user interface elements, to create dynamic, nice-looking, and functional user interfaces. To this aim, the Pliant team recently started the development of the Pliant browser. The Pliant browser consists of a new language (an extension to the *.page* language) for developing Web applications and an HTTP "bridge" for translating the Pliant browser interface to HTTP/Javascript.

For the future, once the additional features provided by the Pliant browser are in place, we plan to give Academia a new makeover on its user interface and features, such as a quiz/survey tool, and added flexibility to course Web page design and "look-and-feel." The style sheet mechanism used by the Pliant browser should greatly facilitate the implementation of a more flexible styling mechanism for Academia. Ultimately, Academia will stand as an example of how end users (teachers) of Pliant tend to become developers of Web-based software solutions. An instructor with little or no programming background can smoothly migrate from a user of a simple system to a programmer of also simple pragmatic systems. The contribution of this framework to students is also noticeable. Student engagement can be improved by online instructional multimedia material, and course online content can be easily tailored to the students' needs.

Since new technologies always afford new roles for teachers as learners and researchers, we also intend to pursue our current research program endeavors, aiming at developing e-learning strategies for the purpose of promoting reflection on teaching and collaborative learning using the Pliant framework. For example, how might teachers/teacher educators use Pliant tools for reflection and research into their classrooms? What are the most critical Pliant design patterns that would optimize their knowledge-building efforts? How will they use that information in their instructional decisions? These challenges are currently under investigation, and our ultimate goal is to produce a solution in the form of an analysis toolkit.

Finally, we would like to initiate a comprehensive evaluation (empirical study) of the use of Academia at both CUSB and Ryerson University. Our targeted audiences are students and staff. We are currently preparing an online questionnaire with the aim of gaining substantial and quantitative-based reactions to the use of Academia as a complement to the already sophisticated Blackboard platform for the purpose of e-learning. Our intention is to measure how far our framework can be useful in delivering and managing online courses.

CONCLUSION

We have described Pliant as a "standalone" and "Web-based" language that encapsulates both the "human" and "computer" levels of thinking and coding programs. This unique privilege makes it an exceptional language, compared to any other existing one. It also demonstrates a higher level of flexibility, reasonable adaptability, and integration, providing for higher software development capabilities and enhancements. These qualities can be exploited to improve the current state-of-the-art e-learning development tools, while meeting educational expectations, economical and time constraints, and human resources limitations. The main advantages of Pliant over other integrated software solutions are high transferability, flexibility, and maintainability. We have also presented Academia, a lightweight courseware application fully implemented in Pliant. When compared to mainstream courseware applications, Academia is surely over-simplified. Finally, a simple, yet concrete example of how Academia was used in a Translation program at CUSB was proposed, along with some insights on the e-learning methodology and pedagogy that were used.

REFERENCES

Blackboard. (2002). *Blackboard Course Management System 5.1.* Retrieved from http://www.blackboard.com/

Booch, G., Rumnaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language user guide* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.

Brusilovsky, P. (2001, October 23-27). WebEx: Learning from examples in a programming course. In W. Fowler & J. Hasebrook (Eds.), *Proceedings of WebNet'2001, the World Conference of the WWW and Internet* (pp. 124-129), Orlando, FL.

Brusilovsky, P., Stock, O., & Strapparava, C. (Eds.). (2000, August). Adaptive hypermedia and adaptive Web-based systems. *Proceedings of the AH 2000 International Conference,* Trento, Italy. Berlin: Springer-Verlag (LNCS 1892).

De Mendez, P.O. (1988). Pliant: Expressive power plus efficiency. *Proceedings of the SALCOM-IT Workshop and Review Meeting,* Barcelona, Spain.

De Mendez, M., De Mendez, P.O., Santos, M.V., & Tonneau, H. (2000). *Pliant documentation.* Retrieved from http://old.fullpliant.org/

E-Workflow. (2003). *The workflow portal.* Retrieved from http://www.e-workflow.org

Hochheiser, H., & Shneiderman, B. (2001). Universal usability statements: Marking the trail for all users. *ACM Interactions, 8*(March-April), 16-18. Retrieved from http://www.acm.org/pubs/citations/journals/interactions/2001-8-2/p16-hochheiser/

Nielsen, J. (1993). *Usability engineering.* Boston: Academic Press.

Nyongwa, M., & Aubin, M.C. (2004). *Plan de développement stratégique: Traduction et langues.* Retrieved from http://www.ustboniface.mb.ca/

Savourel, Y. (2000). XML technologies and the localization process. *Multi-Lingual Computing & Technology, 11*(7).

Tonneau, H., De Mendez, P.O., & Santos, M.V. (1984). *Pliant homepage.* Retrieved from http://pliant.cx

Wang, M. (2000). *Evaluating the usability of Web-based learning tools.* Master's Thesis, Department of Computer Science, University of Victoria, Canada.

WebCT. (2002). *WebCT Course Management System 3.8.* Retrieved from http://www.webct.com

KEY TERMS

Academia: A courseware development Web portal, fully implemented in the Pliant language.

Courseware: Computer software and associated materials designed for educational or training purposes.

E-Learning: The use of network technology to design, deliver, select, administer, and adapt learning activities.

Fullpliant: Name given to the Pliant' operating system.

Pliant: The first efficient, truly extendable, customizable programming language. It is suited both for small scripts and for very large applications, and could be described as a combination of reflexive C, C++, typed Lisp, and clean syntax in a single language.

Software Design: Process of problem solving and planning for a software solution.

Usability: A measure, in our context, of how easy it is to use software to perform prescribed tasks.

ENDNOTES

1 This chapter is an extended version of a preliminary work entitled "Pliant: More Than a Programming Language, a Flexible E-Learning Tool," published in the *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2004* (pp. 505-510). Chesapeake, VA: AACE.

2 Collège Universitaire de Saint-Boniface, University of Manitoba, Winnipeg, Manitoba, Canada.

3 Academia was developed at the Department of Computer Science at Ryerson University, Toronto, Canada.

4 The Co-Op Education and Internship program at Ryerson University is managed by means of a Co-op Web portal available at *http://www.scs.ryerson.ca/~co-op.*

5 This should not be confused with other project of the same name available online at *http://www.pliant.org/.*

6 Meta-programming refers to the ability of Pliant to eliminate the barrier between low-level languages like C and high-level languages like Lisp or Python.

7 Academia is a Pliant-based Web portal. Its architecture is described later in the section entitled "Academia's System-Level Capabilities."

8 Please refer to the section entitled "Why Academia?" for an introduction to this tool.

9 Actors are objects outside of the scope of the system, but that have significant interactions with it.

10 All URLs mentioned in this chapter are fictitious.