Stateless Programming as a Motif for Teaching Computer Science

AVI COHEN
Bar-Ilan University, Israel

With the development of XML Web Services, the Internet could become an integral part of and the basis for teaching computer science and software engineering.
The approach has been applied to a university course for students studying introduction to computer science from the point of view of software development in a stateless, Internet environment. The feedback obtained from the course attests to its success.
This article is an attempt to focus attention on stateless programming and to give this paradigm its appropriate status in computer science studies. The course could provide an alternative to traditional patterns of computer science. The walk-through presented here is a way of demonstrating this new paradigm.
The starting point for the course is the understanding of the Internet environment and the stateless HTTP request, followed by the use and development of XML Web Services integrated with XML and XML schemas, databases and SQL language.
Categories and Subject Descriptors: H.3.5 **[Information Storage and Retrieval]**: Online Information Services-*Web-based services*; H.5.4 **[Information Interfaces and Presentation]**:Hypertext/Hypermedia - *Architectures, theory*; K.3 [**Computing Milieux**]: Computers and Education
General Terms: Design, Experimentation, Human Factors, Languages
Additional Key Words and Phrases: Stateless programming, XML web services XML, XML schema, .net framework, C# programming

1. INTRODUCTION

An introduction to computer science presents the teacher with three different approaches, as suggested by CC2001:

> *"Implementations are an imperative-first approach that uses the traditional imperative paradigm, an objects-first approach that emphasizes early use of objects and object-oriented design, and a functional-first approach that introduces algorithmic concepts in a language with a simple functional syntax, such as Scheme."* [SIGCSEa 2001]

Many colleagues in the field feel that algorithm implementation is critical to the understanding and assimilation of the cognitive basics for teaching computer science. Hence the choice of work environment and computing language is a consequence of teaching computer science:

> *"Most introductory computer science courses have focused primarily on the development of programming skills."* [SIGCSEa 2001]

Author's address: A. Cohen, Bar-Ilan University, Israel; email: Avi@CSIT.org.il

The traditional approach usually suggests that teaching computer science is best performed in closed stand-alone systems and stateful programming.

The approach presented in this article is aimed at the novice student, and includes the elements present in a client-server environment, namely stateless programming (or programming in a stateless environment). This environment is becoming increasingly dominant in the implementation of software systems.

This is an elective course for post-graduate students in the knowledge-management unit in the faculty of humanities. It is an introductory course with no preliminary demands. The students, drawn from different faculties, are interested in expanding their knowledge about computer science in general and the Internet environment in particular. Having taught the course for three years, I can point out that it also attracts some computer science students who wish to expand their knowledge of the Internet environment. These students are familiar with program development and software engineering, so part of the course is relatively easy for them. However, the faculty reached the conclusion that the experienced CS students enjoyed almost no advantage in studying stateless programming. (It is a year-long course of four weekly teaching hours.)

Stateless programming requires that students acquire different cognitive skills throughout their studies. Even if the suggested course is taught to beginners (i.e., CS0, CS1), it can be implemented at any level of computer science (as we saw in the last two years).

The students are not only required to propose an algorithmic solution along with a practical one to a running program, but also to think in terms of where the application operates and when an HTTP request is answered. That is, the next request might not be aware that there had been a previous request; in other words, the non-saving of the state when client-server actions are performed needs to be included. Since a stateless programming approach is not a vital motif in traditional CS0 to CS2 courses, the disconnected, stateless environment, which is an important issue in this course, makes this paradigm the prime motif, one that is vital in designing and planning Internet solutions.

The course also presents a unique approach that contributes to the popularity of the topics that are covered and makes them relevant for a wide variety of computing problems. The declarations that

"Development of a computer science curriculum must be sensitive to changes in technology" [SIGCSEb 2001] and

"Given a field that changes as rapidly as computer science, pedagogical innovation is necessary for continued success" [SIGCSEc 2001]

are working assumptions in a course that combines decentralized client-server problem-solving (i.e., stateless programming) and the following requirements: algorithmic thinking (algorithmic computation, algorithmic efficiency, and resource usage); programming fundamentals (data models, control structures); and computing [SIGCSEc 2001].

In Computing Curricula 2001 (CC2001) at the SIGCSE site, we also found that

"it simply means that there was not a broad consensus that the topic should be required of every student in every computer science degree program" [SIGCSEd 2001]

Indeed, this course does not offer a solution for the entire student population, nor does it include the full program recommended by CC2001. It does seem, however, that with advancing technologies and the basic demands of algorithmic thinking, that integrating this approach in teaching computer science will be increasingly unavoidable.

The proposed approach emphasizes decentralized objects represented by XML and XML schema, which are brought to the client by proxy. The ability to export objects was created technically with the introduction of SOAP (Simple Object Access Protocol) and also implemented by Web services. SOAP is a W3C recommendation and an XML-based messaging protocol used to encode the information in Web service request and response messages.

The export of objects to a distant Web method and reading remote classes adds new requirements on XML Web services, and is an important motif in the advanced stages of the course. The curriculum is based on the need to acquire thinking skills without having to grasp technological complexities, which facilitates our entry into the loosely coupled world without the costly overhead of understanding intricate technologies.

It should be pointed out at this stage that even if efforts are made to bypass "statelessness" by sending the "state" to the client (for instance: cookies, hidden fields, view-state, and state bag) or by preserving the state on the server (by http-application or http-session), the entire process remains "stateless" because it is based on a disconnected situation. This stateless approach is based on the assumption that it is too risky to save the state on the client side and too expensive (in resources) to save on the server side.

The course encompasses six stages: an introduction to the fundamentals of the Web; generating Web pages to thin clients and an introduction to computer science; information management via stateless environments; XML as a hierarchical data contractor; XML Web services; and finally, extending knowledge and a final project.

In planning the course, we considered switching the work environment and language at the halfway stage in order to show that the same paradigm could be applied in different work environments. This idea was rejected in order to avoid the overhead that would be incurred in switching between languages or between environments. In longer courses it might be productive and even interesting, for instance, to start in a .NET environment and at a later stage to switch to an open code environment with Sun's Java 2 Platform, Enterprise Edition (J2EE). The .NET environment was not chosen as the opening example because of its technical advantages, but due to the quantity of documentation about it and the minimal knowledge and expertise required to begin learning it.

> *"This is another area where Microsoft has the benefit of coming to the party late... Microsoft claims that programmers will find it much easer to create web services and to program mobile devices using the new .NET framework"* [Reges 2003]

2. STATELESS PROGRAMMING

Applications on the Web usually run in a stateless environment [Adiscon 2001]. In stateful sessions the operating system retains the application variables, and most of the time the application remains and runs in memory until it has finished the session. The application is always connected to the user-client. If the application connects to a database, the database connection stays open (and may be locked) until the user has finished the cycle of reading the record, sending it to the graphical user interface (GUI), updating the database if needed, and then closing the database.

Stateless programming is an outcome of the stateless, loosely-coupled environment. The algorithm must take into account that it has two parts: a client side and a server or

provider side. Part of the solution will run on the server and part on the local client. The stages of the interaction between server and client always start from a new untrustworthy session; the server may assume that the client is the one to present the stage of algorithm progress.

A simple example illustrates one viewpoint of stateless programming: If we wish to use a loop that will iterate (say, ten thousand times), sum, and print input integers in a traditional stateful environment, we may use a simple for-loop. But this is not obvious when the GUI is a remote browser. The program may send an HTML document to the client with a textbox and submit button. The browser will submit the number (for instance, by http-Get or http-Post) to the server. The following question then arises: How will the program on the server preserve the state of the loop counter and the sum variable? Every time we click the submit button on the browser, the process on the server starts as a new process (for all we know, this may be after rebooting the server) and the server "does not know" why this http request is "knocking" on the server login "door" (is it the first iteration or the *n*th iteration?). Even if we can preserve the state by sessions, view-state, hidden fields, or cookies, or make an XML data object with the Web method in Web service sessions, this process has its limitations, conflicts, and a special design that is not as obvious as the traditional, stateful, simple, for-loop.

Stateless environments are relatively easy to implement by means of resources (such as the operating system, protocols, and hardware) and are thus, in most cases, efficient. Efficiency is usually measured by the number of hits per server up-time, server-time recovery, and client simplicity versus sophistication. The Internet, with its stateless environment, simplicity, and efficiency, has proved itself during the past decade by its vast and rapid expansion throughout the world. Even competitive companies have found simple protocols to overcome differences in their operating systems.

Because recovery is simple, stateless environments are inherently well adapted to unreliable environments; they are also well suited for retrieval-type applications. Their simplicity is good for client-server environments, as opposed to HTTP. In HTTP, the client and the server may be running under different operating systems and may not trust each other completely (i.e., anonymous login or request). Another reason is the number of users connecting to the server. In fact, when the number of clients is far greater than the operating system can handle with an open connection (i.e., stateful), there is an obvious rationale for implementing a stateless environment and maintaining the connection for the request only, but not for the whole session. In this case the operating system holds only a minimum number of session variables. A stateless environment is also suitable for many other situations, and not only for HTTP requests.

A stateless environment is not necessarily transactional [Adiscon 2001]: in a transactional environment, a single transaction (logical group of actions) is either completed fully or not done at all. To the extent that a transaction is formed by a single request and response, many transactional environments are also stateless.

In the Internet environment it is important to make the anonymous untrustworthy application work in a firewall-friendly connection (i.e., text over HTTP request by port 80). This untrustworthy application can use XML to transmit and receive data via SOAP. and the Web services description via WSDL [Christensen et al. 2001]. The Web Services Description Language (WSDL) is an XML-formatted language used to describe Web service capabilities as collections of communication endpoints capable of exchanging messages.

We have reached an era when a stateless environment empowered by XML and XML Web services is important for some computer science courses. It is obvious that XML

and XML schemas are very important in this course when discussing variables, classes, and instantiated objects .

Throughout the course, it was not clear whether the students assimilated the state problems between client and server. We encountered questions like "Where is this code running? Does this code execute again when the data is posted back? What is the raising events order on the next post back? Do we really need to use SOAP if we can use Post or Get methods? Where is the object really running after the application imported WSDL? How do we increment the index in every post back? "

3. CHOOSING THE TECHNOLOGY

Just as the choice among C, Pascal, Java, C#, or C++ does not determine algorithm principles, neither does the choice of technology change pedagogical principles in a loosely coupled, stateless environment. Apparently, software companies compete for the strongest developer tool and the best runtime implementation of Web services. A review of standard products (June 2005) indicates that there are currently no significant differences among the offerings of the various companies (and these differences are insignificant for the course proposed in this article).

The .NET technology was chosen for the course for two main reasons (rather than for its technical advantages): the first (as stated above) is its friendly teaching environment for the beginning student.

> *"most observers would agree that in this department Microsoft has the edge with its Visual Studio .NET, the Microsoft Developers' Network (MSDN) and the other tools and support services it offers."* [Rubens 2003]

The second reason is that we had to consider the operating system that the novices had already installed on their personal and laptop computers. Installing .NET and using the Internet information server seems to be part of the same operating system, and thus easy to do at this introductory stage. Moreover, we assume that some students will not continue in the computer science track.

We assume that once the students have acquired the principles, the switch to another technology is technical and to some degree intellectual, but in no way fundamental. After teaching this course for three years, the validity of this assumption was confirmed when we interviewed one of the graduate students about his actual work in industry. Henceforth, we will concentrate on this technology and reduce comparisons to other technologies to a minimum.

Our internal conflict was between C# and VB.NET. In .NET technology the language loses much of the significance it had in the traditional approach. The .NET framework consists of two main parts: the Common Language Runtime (CLR) and a unified, hierarchical class library. Each of the .NET languages is compiled in relation to the Intermediate Language (IL), which is then converted to native code and executed on the CLR. In other words, the traditional VB has been upgraded to a full object-oriented multithreaded language like any other .NET language. Hence in most cases the choice is semantic. Some teachers might choose VB.NET, which supports controls and language similar to traditional VB; others might choose C# in order to use pointers and a language that is similar to Java. The switch from Java to C# is simple, which makes the switch from the technology we have chosen to another one even easier [Reges 2003]. In the first two years, VB.NET was the chosen language; this year (the 2004-2005 academic year) I felt it would be refreshing to use C#, and found no difficulty switching between the two. During the first two years, VB.NET enjoyed an advantage in the beginners' literature.

This year, after conducting an extensive survey on C# and VB.NET literature, it appears that C# literature has an advantage over VB.NET.

The technology used in this course allows the developer to enjoy both worlds. The framework (similar to the JAVA virtual machine) is at the bottom, above the operating system, and the developer's friendly interface is on top. This means that, on the one hand, the student can enjoy a friendly interface and, on the other, managed code. Explaining the operating system and the framework is up to the teacher, and should be consistent with the student population and the time allocated to the explanation.

So far we have made two choices, the more difficult one that of selecting the technology, and the other – within the technology – of choosing a language.

4. COURSE PATH

4.1 Introduction to the Fundamentals of the Web

Since the course is based on a loosely coupled, disconnected, and stateless environment, that is to say, the Internet environment, it is important to teach the students the basics of Internet structure. This is the stage during which the student should come to understand Internet principles and the role of HTML objects as a way of presenting information. At this stage the four principles of content presentation that constitute the basis for further study are hyperlinks, tables, text and picture presentations and, most important, forms, including the post and get methods. From the beginning, the student needs to internalize the principles of DOM (document object model), which is the specification of how objects in a Web page such as text, images, headers, links are represented, as well as the browser's information-presentation system. Although the term *client-server* was imported from mainframe terminology, we use it for N-tiers (as far as we are concerned, the client requests the information). This is true for every HTTP requester, even if the requester is a middle tier.

The DHTML (Dynamic HTML) part is minimal; it is only used to clarify the concept and its integration in HTML (in an HTML line and not as a separate script) – for instance, to change a picture on mouse-over without calling a function or to change text properties on mouse-click. At this stage of the course, we assume that the student does not know what functions or programs are. (The issue of CSS, cascading style sheets, is not an essential part of the course.) The teacher can exercise discretion in mentioning the topic or in using simple examples; some institutions might develop courses that focus on this area, but these are totally different courses.

Some teachers might opt to discuss the evolution of the Web via the chosen technology and traditional remote procedure call [Shohoud 2003]. The proposed course offers a very brief summary, clarifying traditional pre-NET or JSF terms (JavaServer Faces technology); it emphasizes that these terms are not "in our neighborhood," and are not relevant. The past is behind us. The goal is to use existing tools to internalize principles, not to focus on the tools. Any simple text editor can serve this part of the teaching.

This phase of the course was relatively short; the general impression was that most students' previous knowledge is adequate to grasp the explanations about the essentials of the Internet.

4.2 Generating Web Pages to Thin Clients and Introduction to Computer Science

The first stage was mainly devoted to familiarizing the students with a "thin client." The server returned HTML pages without generating the program that creates HTML. The second stage, described in this section, involves the student moving to the server. The

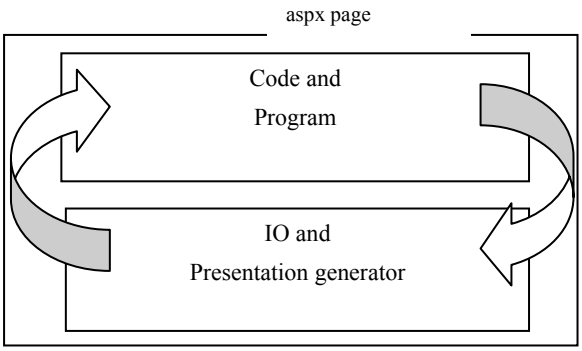

Fig. 1. ASPX document and code presentation.

student goes from ready HTML files (that are sent over HTTP request to the client) to files that are created and generated by a high-level language program on the server.

This is where traditional relations with computer science begin, via the creation of Web pages in the selected language. The students learn the basic principles in the programming fundamentals course (SIGCSE 2001) and other aspects of CS that correspond to teaching a programming language.

> *"Fluency in a programming language is prerequisite to the study of most of computer science. We believe that undergraduate computer science programs must teach students how to use at least one programming language well…. As a result, this area includes units on fundamental programming concepts, basic data structures and algorithmic processes. These units, however, by no means cover the full range of programming knowledge that a computer science undergraduate must know."* [SIGCSE 2001]

Regarding the environment, the student learns the following simple structures of the presentation and code sections via ASPX documents. ASPX documents (or pages) are active server pages that run on .NET technology. Like PHP and Perl pages, the ASPX pages generate HTML documents. ASPX documents are separated into two main sections: the upper section is the script (program) and the lower one is the presentation generator (HTML or HTML-like). We can view these two sections as a presentation generated for the client, and a "program" that transfers the results to the presentation generator as shown in Figure 1.

At this stage the student learns to recognize the various server controls that create the presentation (Web) pages. Developers in this technology are faced with a number of possibilities, ranging from Web server controls to HTML controls. The most suitable method for our purposes is Web server control, characterized by the opening tag prefix "<asp:TagName>". In many cases the "TagName" may be a traditional HTML tag. A Web server control is a built-in .NET class that enables manipulation on data presentation. The serialized class on the server actually generates the HTML tags that return as an http answer.

An example of the explanation about the <asp:TagName> is to create a loop that constructs a table. Following this, the same table is created by a Web form that eliminates the need for generating the table by code. Feedback obtained during the past two years shows that this example clarifies the principles of this technology very well, and even the

principles of the next generations of this or alternative technologies, e.g., the Sun's JavaServer Faces (JSF) technology.

In this course, students who were not familiar with traditional active server pages (ASPs) felt comfortable with explanations for the need to move back and forth with the state in hidden fields and the use of viewstate. The course avoids explanation of render code generation that typifies traditional ASP pages, which have returned the code-developing world to the prestructured programming era. Because we address students who are studying computer science and algorithmic thinking from step one, we regard the ASPX server pages as an input-output generating system in a managed-code environment. As an outcome of the stateless environment, some traditional patterns of basic programming had to be thought through via a different approach (for instance, an exercise in stack simulation).

The existing approach allows us to select from a number of currently developing tools, the simplest being any plain text editor. Another suitable dedicated tool, which we used until the 2004-2005 academic year (it is also free software), is the ASP.NET Web Matrix developer's tool [ASP.NET 2004]. To some extent, the logic of this course can be seen there (as shown in Figure 1). The Web Matrix is similar to the next generation of editors, called Visual Studio Web Developer 2005 (which we will use next year) is a lightweight (a few megabytes), free WYSIWYG editor that is enhanced by many wizards. The Web Developer is much more advanced and user-friendly than the traditional Web Matrix developer's editor, especially for novices. It facilitates the manipulation of Web controls and script writing, and even has a built-in server that facilitates browsing ASPX server pages. Like the Web Developer, the Web Matrix creates ASP.NET (i.e., ASPX) server pages using the inline approach rather than the code-behind approach. In a code-behind approach, codes that are responsible for different purposes (e.g., a presentation code versus its code-behind, i.e., the methods) are broken down into different files; this concept is especially designed for large projects. When everything is on the same server page, it is called an inline approach. Like the Web Matrix, the Web Developer aims to focus on those tasks that fulfill 80% of the requirements for building ASP.NET applications [Homer and Sussman 2003]. Teachers can also use other free inline *open source* tools such as SharpDevelop [iCShapCode 2004].

Even if, due to the Web Matrix, we ignore IIS (Internet Information Service), as mentioned in the previous paragraph, IIS is strongly recommended in building and surfing a virtual directory as a real simulation of the Web. In the advanced stages of the course, the students learn how to use the Web Matrix class browser.

At the present stage, the significant benefit of not using the code-behind approach, despite its advantages (as presented in Visual Studio.NET), is connected to how a project is constructed. Every project is a solution that breaks up files and builds an entire system. The methodology of code-behind and the creation of files that are sometimes deeply hidden in the Solution Explorer are not relevant at this stage (and might frighten novices). The "Solution Explorer" mentioned in VS is a kind of file-manager that shows all the directories and files in the project (which in .NET technology is called a "Solution"). In the past year this approach has been justified by the next generations of Visual Studio, and Web Developer.

The terms *object* and *inheritance* are assimilated through the "algorithm" and "algorithmic problem" approach, which in many ways is reminiscent of the traditional approach. Objects and classes will become familiar at a later stage (as will Web Services and a certain amount of server-programmed controls and business objects).

At this point we need to ascertain that the student has internalized basic (and also traditional) programming, program flow, and critical thinking in a stateless environment with the approach supplied by the environment itself. The next stage is environment development and the use of data from databases. To this end, we widen the students' knowledge of SQL and integration of queries in ADO.NET. In the technology we present, ActiveX Data Object .NET (ADO.NET) is a high-level interface for data objects that can be used to access most types of data, including Web pages, spreadsheets, databases, and other documents.

4.3 Information Management via Stateless Environment

Deepening the knowledge of databases is linked to teaching SQL, which includes an explanation of relational and hierarchical data structures and instructions that are relevant to ensure a continuous flow in an ongoing course according some CC2001 objectives (depending on the length of the course):

> *"Information Management (IM) plays a critical role in almost all areas where computers are used. This area includes the capture, digitization, representation, organization, transformation, and presentation of information; algorithms for efficient and effective access and updating of stored information, data modelling and abstraction, and physical file storage techniques. It also encompasses information security, privacy, integrity, and protection in a shared environment*." [SIGCSEf 2001]

At this phase in the course, we move from a client-server situation (two-tier) to three tiers and more. The server, in this case, becomes the client who receives information from the database (even if the database is actually on the same computer). The technology enables use of a DataSet, which is an in-memory data structure. The DataSet can import records in a database structure, including table relationships, and perform complex updating and record manipulating processes. Any database can be chosen; for this course we chose Access, the simplest and most familiar one.

It is important to clarify the additional problems faced by databases that operate in an Internet environment (for instance, overload, security, foreign platforms, and a longer record lock-time in a stateless process).

When many clients are connected to a server, it is customary to reduce loads by duplicating the server. Where a database is involved, duplication can cause further problems and data discrepancies. With our chosen technology, we can reduce the load by transferring a DataSet. With its accessibility to databases [Plourde 2003], the DataSet enables the transfer of relevant data, as well as update the database and learn whether data has been modified since the last visit (to whoever has the .NET framework and can receive DataSets). Furthermore, the data is transferred by an XML document, which makes the transferred data readable and friendly. Course discussion should also relate to the transfer of a large database and the viability of using DataSet versus DataReader, that is to say, the gain or loss of the server's processing abilities versus the capacity to send a large amount of data. In the technology that is presented, DataReader or DataSet are alternative methods for importing data from a database. The technology we have chosen allows the use of standard Web Server controls in order to learn the principles for this phase. Any server control, for example, Repeater, DataList, DataGrid, or GridView (in Web Developer using .NET framework 2.0), can demonstrate the data transfer methods in the proposed concept. The DataGrid Web server control was selected for this course (followed next year by GridView). DataGrid, like the GridView Web control, has by far the most versatile (but the least flexible) features [Mitchell 2003]. This option ensured

that we did not digress into manipulating HTML server controls joined to the Repeater server control. We assumed that the students did not require further elaboration on the DataList server control.

The subject of transferring data leads us to a discussion of XML and the relocation of data from the server to the client (who may be a provider).

4.3 XML as Hierarchical Data Contractor

This article, like the course, discusses two aspects of XML. The first is a definition of data objects and object schemas, while the second is a description of Web (or remote) methods. This section deals with the former, while the following section (which relates to Web services) deals with the latter. CC2001 (IM12) includes familiarity with HTML, whose structure is similar to XML syntax. Whereas HTML is meant to present information, XML is meant to organize it.

XML enables friendly transfer of data between various platforms, since it is a text file (even if an element contains binary data) that transfers well at port 80 over HTTP. The three important components in teaching the XML structure are elements, attributes, and namespaces [Skonnard 2003]. Our view is that delving deeply into XML schema-defining types in a namespace is only appropriate for advanced students; hence in the present course we explained the terms only briefly.

XML restores the ability to deal with the transfer of hierarchical information, together with the ability to present relational information with the help of XSD (XML schema definition). XSD files can be demonstrated by Visual Studio.NET to explain the definition of elements and attributes, as recommended by W3C.

The best analogy is that XML schemas are like classes and XML documents are like objects [Gonzalez 2003]. Typically, a class uses a template for objects to be instantiated, which is what XML schemas attempt to do with XML documents.

The DataSet mentioned earlier was designed to abstract data independently of the actual data source. The student can see this by populating the DataSets with XML documents versus SQL queries. Thus, it is also important to show parallel properties in the treatment of databases via XML documents and DataSets.

It is important to use programming instructions, even when only mentioning other XML representation methods as CSS in a browser. It appears that familiarity with high-level languages while dealing with information presentation through XML contributes to a better understanding of both topics.

The dilemma of when to teach XML should be mentioned here. The first option is to teach it prior to DataSets and ADO.NET, while the second is to do so via Web services, the transfer of objects, and description of methods. The natural evolution of the Web tipped the scales. The attribution of XML as representing a data object led us to present XML at this point and to define its role. Defining the description of functions, which is more suitable for Web services, will be explained when we deal with Web services. At that stage we can complete the picture of how XML functions and XML serialization/deserialization.

4.4 XML Web Services

After learning the importance of working in a stateless environment and having gained the ability to exchange messages in a loosely coupled environment, students now possesses a solid foundation to start learning XML Web services. From the point of view of this course, introduction to XML Web services is the basis for further computer science studies.

The present stage of XML Web Services has four main substages. The first is forming a Web service on the provider side by creating an ASMX file. In the chosen technology, an ASMX file is one that has an asmx extension that contains the components (class members, Web classes and methods) that enable invoking the remote Web methods by the serialization process. In this phase, the ASMX file was created with the Web Matrix developer tool (from next year on, the Web Developer). The advantage of using Web Matrix (or Web Developer) is in its "clean" code, free of Visual Studio.NET additions; it is a comprehensive tool for large and complex projects.

The second substage is the presentation and discussion on the client side, in which Web methods are exposed and SOAP, XML Schemas and WSDL, which describe the Web Methods, are presented. It is important at this stage to clarify the ability, which is mainly due to the SOAP message, to transfer objects to the Web method (requested via HTTP post). This is an innovation that is opposed to the traditional use of post and get. By discussing the use and transfer of objects and by presenting Web methods, students become aware of the serialization process and the building by proxy that enables the program to invoke Web methods.

XML serialization allows an object's public get/set properties, methods, and fields to be reduced to an XML document that describes the publicly visible state of the object. This method serializes only public properties, methods, and fields. Private data and pointers will not be persist, so that XML serialization will not fully match the original object in all cases. This is not the place to show the limits of automatic ASMX-generated WSDL with read/write-only properties or overloading; this topic belongs to the next course and another article.

During the course we were very careful to adhere to W3C recommendations and to avoid discussions of developments that were yet to happen.

The third substage is the development of the "thin client" side, namely the Web server that uses the Web methods supplied by a third-party provider; the server supplies the browser with the presentation on the thin client's side. This is not a special stage but a combination of elements in ASPX pages created on the server (this stage was learned previously). As a result of questions and continuing confusion on the part of students, the use of Web services for thin clients by a different server should be clarified. The simplicity of Web Matrix may necessitate the presentation and discussion of command-line instructions (that is to say, CSC.exe and WSDL.exe command lines) or the use of the Web Service Proxy Generator automatic tool.

The fourth substage brings us to the more complex world of the "rich client." Whereas until now we have used the Web Matrix tool, for the rich client we introduced the Visual Studio.NET tool. The students learned how to create local programs (based in part on codes learned in the second phase of the course) and learned object-oriented programming. Objects were an integral topic throughout the course, so that when students were required to plan and build classes, they did not experience any difficulties. It should be noted that at this juncture serialization and the entire process of checking the WSDL were hidden from the developer.

One of the exercises the students found exciting is the tic-tac-toe game over the net (two players in different locations in the lab or at home). This game needs Web services to manipulate the database and to update the state at the client side. To make it easy to focus on the Web services, we used a loop that checks the Web service to simulate the game state. This simplicity can be upgraded in advanced courses by using processes, service-oriented architecture (SOA), and advanced software knowledge.

4.5 Final Project

The final phase (by then the student is already familiar with the environment) is one of expansion and the practice of phase five, combined with a final project. The final phase contains Web services that include a wider range of applications and a small amount of the business components.

At this stage the project is intended to give a sense of control, as indeed it did. The final stages of the course include a review of additional possibilities for connecting to cellular phones and opened by Web services.

5. SUMMARY

As this course has progressed, it has taken into account the need to place each teaching element in its proper location in order to create a unique logical continuum. The integration of the stages in the course and in the work environment are largely responsible for assimilating the general ideas underlying this course. The timetable for each phase depends mainly on how far the teacher chooses to go and on the level of learning that is desired. In this course, pedagogic prohibitions against the creation of islands of ignorance and sporadic jumps from one subject to another are taken seriously; but on the positive side, the assimilation of basic topics of computer science and stateless programming are required.

The authors of CC2001 claim that: "We are convinced that no one-size-fits-all approach will succeed in all institutions" [SIGCSEc 2001]. Each university or college should adjust the approach presented here to its student population and required depth of knowledge. The course has been very popular with the students, who feel that they are combining innovative and relevant stateless concepts with the traditional approach of computer science.

The 2004-2005 academic year is the third one in which this course is offered. From random discussions with graduate students, it seems that they are enthusiastic about implementing the skills acquired in this course in continuing studies. The influence of this stateless programming approach on other courses has yet to be investigated. This course or a similar course should also be taught with alternative technology, which will facilitate comparison as well as an assessment of the assumptions underlying the stateless programming paradigm.

REFERENCES

ASP.NET Web site. 2004. http://www.asp.net (access date March 24, 2004).

ADISCON GmbH site. 2001. Programming in stateless environments. Adiscon website. http://www.adiscon.com/IIS/isapi005.htm (access date March 14, 2003).

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. 2001. Web Services Description Language (WSDL) 1.1. Adiscon web site. http://www.w3.org/TR/wsdl (access date May 9, 2003).

GONZALEZ, J. 2003. Building an XML and XSD schema validation tool. 15-second website. http://www.15seconds.com/issue/021022.htm (access date May 3, 2003).

HOMER, A. AND SUSSMAN, S. 2002. *Inside ASP.NET Web Matrix*. Wrox Press Ltd.

iCsharpCode Site. 2004. http://www.icsharpcode.net/ (access date Feb. 10, 2005).

MITCHELL, S. 2003. Deciding when to use the DataGrid, DataList or Repeater. MSDN Library. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/aspnet-whenusedatawebcontrols.asp (access date Sept. 10, 2003).

PLOURDE, W. 2003. Handling concurrency issues in .NET. 15-second website. http://www.15seconds.com/issue/030604.htm. (access date Oct. 5, 2003).

REGES, S. 2003. Can C# replace Java in CS1 and CS2? p. 5. Univ. of Arizona, Computer Science Dept. site. http://www.cs.arizona.edu/~reges/sigcse/csharp.pdf. (access date June 7, 2003).

RUBENS, P. 2003. How do J2EE and .NET measure up. *ASPNews.* http://www.aspnews.com/trends/article/0,2350,9921_2200571,00.html. (access date May 17, 2003).

SKONNARD, A. 2003. Understanding XML namespaces. *MSDN Library*. http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xml_namespaces.asp. (access date Aug. 28, 2003).

SHOHOUD, Y. 2003. RPC/literal and freedom of choice. *MSDN Library*. http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/rpc_literal.asp. (access date Oct. 11, 2003).

SIGCSE a. 2001. Principles, computing curricula 2001. Ch. 7, Sect. 7.2, *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/cs-introductory-courses.html. (access date April 4, 2003).

SIGCSE b. 2001. Principles, computing curricula 2001. Ch. 4, Para. 4, *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/cs-principles.html [access date: April 10, 2003]

SIGCSE c. 2001. Principles, computing curricula 2001. Ch. 7, Sect. 7.1, *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/cs-principles.html. (access date April 10, 2003).

SIGCSE d. 2001. Principles, computing curricula 2001. Ch. 5, Sect. 5.1.1, *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/cs-principles.html. (access date April 12, 2003).

SIGCSE e. 2001. Principles, computing curricula 2001. Programming fundamentals (PF), *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/PF.html. (access date April 15, 2003).

SIGCSE f. 2001. Principles, computing curricula 2001. Information management (IM), *Computer Science Volume*. http://www.acm.org/sigcse/cc2001/IM.html. (access date April 14, 2003).

Received March 2005; accepted June 2005