A Funny Thing Happened on the Way to the Form: Using Game Development and Web Services in an Emerging Technology Course

Randy Connolly
Dept. of Computer Science & Information Systems
Mount Royal College
Calgary, AB, T3E 6K6, Canada
rconnolly@mtroyal.ca

ABSTRACT

This paper presents the results of an emerging technology course devoted to web services and games development. The paper defines web services and service-oriented architectures in general, covers the rationalization for the approach taken in the course, and describes the scope and design of the game project. It also suggests how web services and/or game development can be integrated into an upper-level emerging technology course, and analyzes the students' (and the instructor's) learning experience in the course.

Keywords: emerging technology, web services, service-oriented computing, game development, .NET Framework

Old situations
New complications …
Passions and potions
Constant commotions
Something for everyone –
A comedy tonight.
– *A Funny Thing Happened On the Way to the Forum (*Sondheim, 1962).

1. INTRODUCTION

Many information systems programs offer some type of emerging technology course. The ACM IS 2002 Model Curriculum for instance, has as one of its ten logical course specifications, a course devoted to "design and implementation within an emerging environment." According to the Model Curriculum, such a course should focus on "implementation in emerging distributed computing environments using traditional and contemporary development methodologies" and should expect the student "to implement a project that spans the scope of the previous courses" (Gorgone et al, 2002). This paper presents the recent results (Winter 2004) of an emerging technology course that is part of the Applied Degree in Computer Information Systems and Business at Mount Royal College in Calgary. In this course, students were exposed to two separate, but combinable technologies: games development using C# and Windows Forms and service-oriented computing via SOAP-based web services. This paper reports on the "Old situations/New complications" encountered by the instructor and students in the course. It provides a definition and overview of web services and service-oriented architectures in general, covers the rationalization for the approach taken in the course, and describes the scope and design of the game project. It also suggests how games development and web services can be integrated into an upper-level emerging technology course, and offers some analysis of the students' (and

the instructor's) learning experience in this course.

2. SERVICE-ORIENTED COMPUTING (SOC) AND WEB SERVICES

Service-oriented computing (also known as service-oriented architectures or SOA) in general and web services in particular, is perhaps the hottest – or certainly the most hyped – new technology within the software application world. IBM (Channabasavaiah, 2004) claims that SOC is "the next evolutionary step in software" and that

> … after all the hype has subsided, and all the inflated expectations have returned to reality, you will find that a SOA, at least for now, is the best foundation upon which an IT organization can take its existing assets into the future as well as build its new application systems.

What is service-oriented computing and how does it relate to web services? SOC is the "computing paradigm that utilities services as fundamental elements for developing applications" (Papazoglou, 2003). The fundamental use of services can dramatically alter the way one architects an application. Such an approach results in "an application architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes" (Channabasavaiah, 2004). Thus SOC is "an architectural style whose goal is to achieve loose coupling among interacting software agents" (He, 2003). Figure 1 illustrates the difference between the traditional integrated application and the new SOC application (adopted from Chatterjee, 2003).

The rationale behind this new design paradigm is one that will be familiar to most computing practitioners with some experience in the enterprise: namely, how to best deal with the twin problems of integration complexity and reuse. Due to corporate mergers, longer-lived legacy applications, and the need to integrate with the Internet, application integration has become a major priority of IT organizations. The principal difficulty with application integration however is the sheer number of interfaces

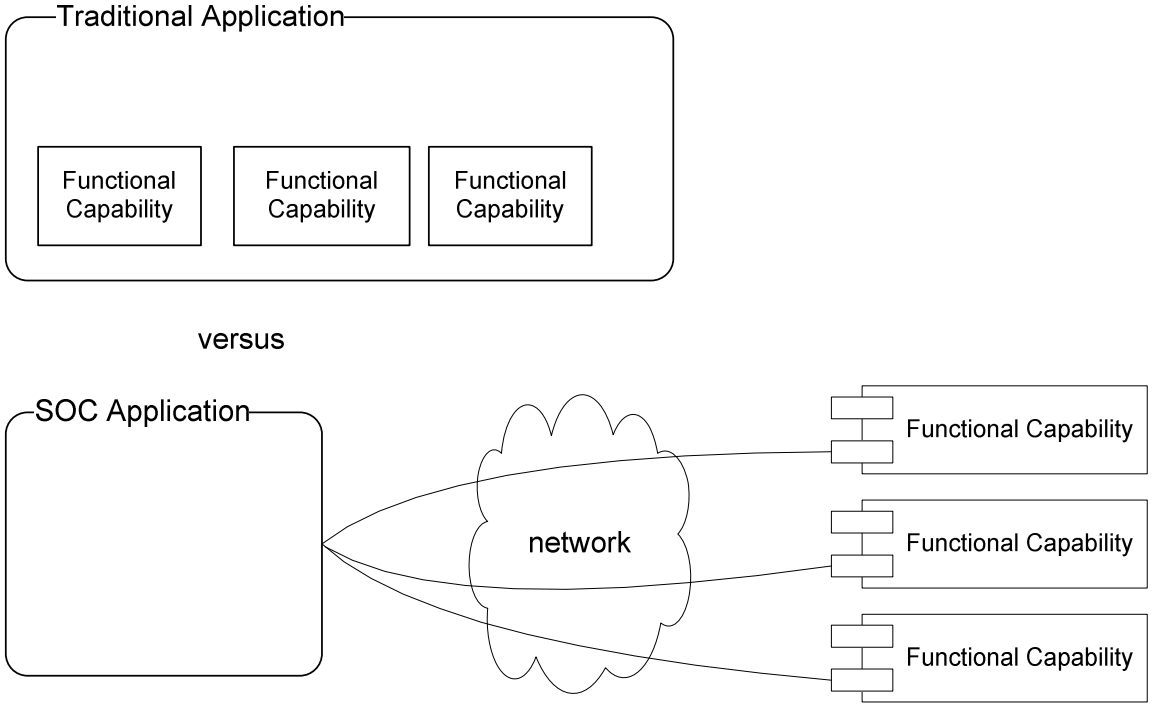


Figure 1

required. If there are *n* systems that need to be directly integrated with each other, then *n*(n-1)* interfaces will be necessary, and each new system will require an additional *n*2* interfaces (Channabasavaiah, 2003). As well, since these systems are typically not interopable, expensive middleware systems or complex object messaging solutions (such as RMI, CORBA, and DCOM) are necessary to implement these interfaces.

Service-oriented computing potentially provides a more palatable solution to these integration problems. But what is a service? A service is simply "a unit of work done by a service provider to achieve desired end results for a service consumer" (He, 2003). It is a self-describing, self-contained, open interface to piece of functionality (Cerami, 2002); that is, it should provide a platform-independent interface contract that can be dynamically located and invoked, and which contains no state (Hashimi, 2003). Since these services can then be offered by either different systems within an enterprise as well as by different enterprises, they "provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration" (Papazoglou, 2003). SOC promises then a "nirvana, in which discrete channels of business logic become reusable, interchangeable parts that can be strung together into business processes with almost no development cost" (Knorr, 2003).

While SOC as a concept predates the technology of web services, it was the standardization provided by web services that made SOC viable. HTTP and XML are used both to publish and consume a web service. The two additional platform-independent XML-based protocols of SOAP and WSDL constitute the basis of web services. WSDL (Web Services Description Language) describes the operations provided by a service; that is, it documents and describes the data types and signatures of the operations. SOAP (Simple Object Access Protocol) encodes the service invocations and their return values within a HTTP header (see figure 2).

3. USING THE .NET FRAMEWORK

While web services are by design platform-independent, some platforms make it easier to construct and consume them. Microsoft's .NET Framework makes it quite painless to work with web services and for this reason (as well as the fact that this semester's students had already taken ASP.NET and C# from the same instructor in the previous semester) was chosen as the development environment for the course. In the first lab, students were able to consume a language translation web service, as well as consume Amazon.COM's web service (which makes all of Amazon's data and functionality available). Creating a web service was not much more difficult. In the next lab, students created a credit card validation service that made use of data within a SQL Server database. Given the relative ease of working with web services in the .NET Framework, something else was required to fill up the fifteen weeks of the semester!

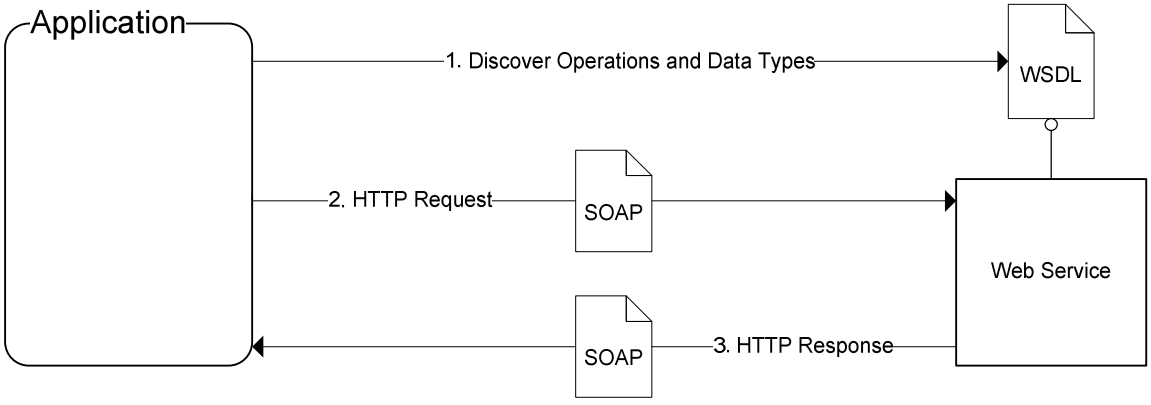


Figure 2

In our program, students are exposed to a variety of application development environments. Students take two courses devoted to creating web applications, three courses to teach programming (from structured to object-oriented) using Java, and three courses teaching databases and rapid application development using Microsoft Access or Oracle Forms. One perceived lacuna in the students' education is that they never create native Windows client applications. For this reason, it was decided to "fill the time up" in the emerging technology course with Windows Forms development, which is the .NET replacement for the C++/MFC approach to native Windows development. The plan was to teach the basics of Windows Forms development, move back into web services, and then have students create some type of desktop client for consuming each other's web services.

4. DEVELOPING A GAME (THE PLAN)

But a funny thing happened on the way to the Form. The initial plan for the course project was a typical business application, albeit distributed via web services. All such applications tend to have a similar workflow: retrieve data, transform and present data, validate changes to data, and save data. In two previous courses, the students had already covered several typical data access patterns and were reasonably familiar with layering and architecting the typical business application using classes. Since the students were using Visual Studio, they were able to learn the basics of creating Windows applications within two weeks. During this time, it became quite apparent that what really captured their interest was drawing graphics and interacting directly with the mouse and keyboard. Seeking to maintain and use this interest, I decided to change the course project from a business application using web services to a graphical game using web services.

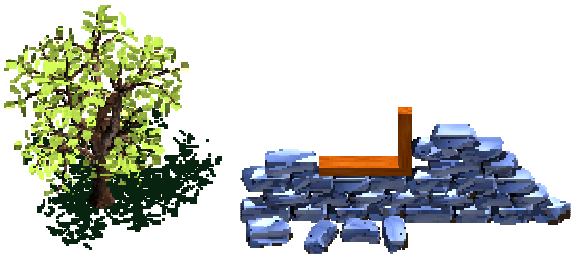
My hope was that through a game project the students would be more motivated to learn. Within the field of education, there is "an abundance of literature to support the use of games as tools that help learners" (Mungai, 2002). Within the context of computer science, a variety of researchers have found game assignments to be helpful for teaching and motivating introductory programming students (see, for instance, Be cker, 2001; Giguette, 2003). As well, Jones (2000) has noted that games can provide "an extremely project-oriented, upper-division course to exercise and enhance the programming and problem-solving skills of advanced students." Another motivation for switching to a game project was relevance. Given that the students do not have a great deal of real-world work experience, they may find it difficult to appreciate the typical integration problems that web services address. It was hoped that a game project would be a more familiar context to them and hence would better communicate the distributed nature of web services and their integrating role. The remainder of the paper will describe the game project, problems (and opportunities) encountered, and how web services were eventually integrated into the project.

The game project was a "simple" role-playing game. Students worked in pairs to create a game in which a player (in the role of a barbarian, knight, wizard, or ninja, each with its own unique statistics) navigates a multi-screen map and fights monsters based on a configurable combat system. For simplicity sake, students used GDI+ rather than DirectX for drawing graphics. Various additional custom controls had to be created to handle status messages, the character's state, and the character's position in the game world. As little game information (e.g., map, actors, world, etc.) as possible was contained within the code; instead this information had to be contained in XML files.

The students were provided with a variety of royalty-free graphical resources (figure 3 shows some samples). These included several hundred tile files (organized into sets) to construct maps, static and animated item images for placement on the map, as well as over a hundred animation strip images for player and monster actors. Each monster or player actor had four direction facings (north, east, south, and west) and five states (paused, walking, attacking, being hit, and dying) that had to be animated.

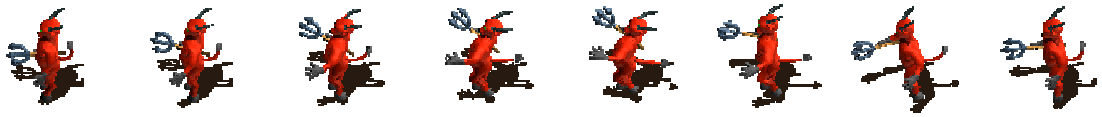
In the course labs, students were introduced to the following: GDI+ development, creating custom controls, parsing XML, and

sample tile images



sample item images

sample actor animation strip image (attacking west)



sample monster animation strip image (attacking west)

Figure 3

working with timers. In one of the labs, students constructed a simple sprite (an independent animated object) as a means of demonstrating how to use an event-based timer as a first step in learning multi-threaded programming. Upon this foundation, students were to unproblematically construct this game (early March deadline) as a first step in the eventual web service-enabled game. But "New complications/Constant commotions" were indeed encountered along the way!

5. DEVELOPING A GAME (THE REALITY)

While students were very excited by the project *before* they began it, as they worked on it, they found it to be perhaps the most difficult assignment they had encountered in the program. Students could not simply replicate the typical business application process – read, display, validate, and write data – and its typical architecture (presentation layer, business layer, data layer). Instead, students were forced to construct their own architecture (for the world and its maps, for the actors, for the sprites, and for the combat system) and process (e.g., when should the map files be read, when should the images be read for the actors, should all the images be stored in memory, etc).

In previous application development courses, I have found it helpful to introduce one or two of the standard Gang of Four design patterns (Gamma, 1995). In my object-oriented development course that I also teach, I have often struggled to find appropriate contexts for teaching design patterns. This project, in contrast, was a goldmine of potential pattern usage; it provided a context that allowed many of the students to

grasp and comprehend both the utility and the beauty of these patterns. In the context of the game project, I was able to demonstrate the practical usefulness of the following patterns: Singleton (for creating a single repository of all images), Observer (for handling the game events caused by the actors which needed to be handled by the game environment), Mediator (for coordination between different user controls), Factory (creating GDI+ images based on tile keys), State (for handling the actor's state), Strategy (for handling the run-time configurable combat systems), and Command (for handling different user-specifiable game actions).

Yet despite of (or perhaps because of) these helpful patterns, the student feedback was not uniformly positive. Several complained that the game required them "to think constantly." Other students commented that they "had to use all the stuff [knowledge presumably] from previous courses." The student experience appears to back up Jones's (2000) belief that the integration of concepts and techniques required to design and build computer games covers many of the topics offered in an undergraduate computer science curriculum, allowing students concrete application of much of the theory, concepts, and skills they have been exposed to.

The conceptual difficulty of game development for the students was reflected in the eventual marks. While the average was a respectable 68%, the marks tended to be either in the A range (small majority) or in the D range, clearly both a victory and a defeat from an instructor's standpoint. The lower marked projects tended to have non-functioning combat systems (which required two sets of timers and thus the management of three execution threads). Nonetheless, the quality of several of the student's games was very rewarding (see figure 4).

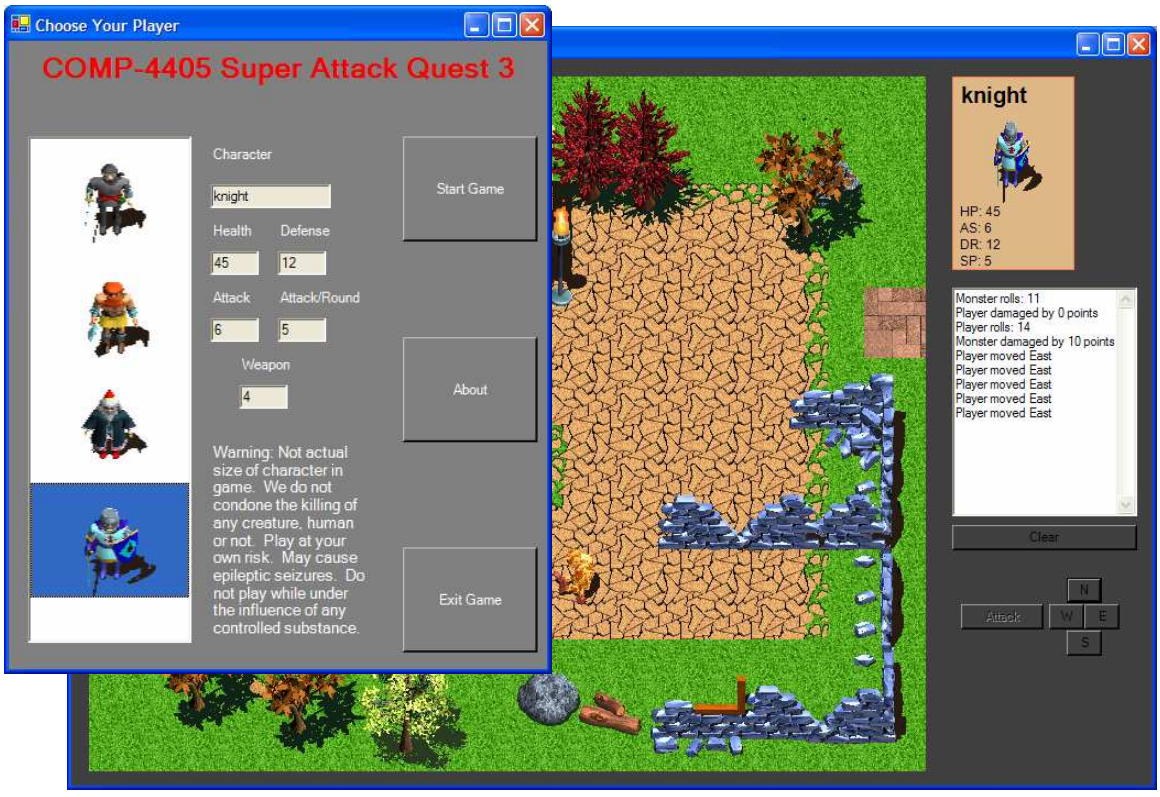


Figure 4

6. WEB SERVICE INTEGRATION INTO GAME

The original plan for web service integration in the game project was to pull certain functionalities out of the game and place them within web services. World definition (and their maps) would be obtained by the game client from any of the students' game web services. Monster and character statistics and all combat result calculations would also be pulled from the web service. However, due to the fact that a large minority of students were unsuccessful in implementing all the functionality of the first iteration of the game, I felt compelled out of fairness to provide an alternate second part of the project (I could have provided them with my working version of the first part, but did not because I felt it would compromise next year's delivery of the course). Rather than extending the game, the students created an editor that could load, edit, and save XML isometric-tile maps. The editor could load the map from a file or from a web service. Each student had to also create and publish a web service which provided an XML stream that could be consumed by the other students' editors. Rather than using UDDI or DISCO for the discovery of the services (due to security-related problems with the lab), the students were provided with a URL of another web service that returned a list of available student map services and their URLs. Figure 5 shows a screenshot of a finished sample editor (which shows a map in the process of being defined).

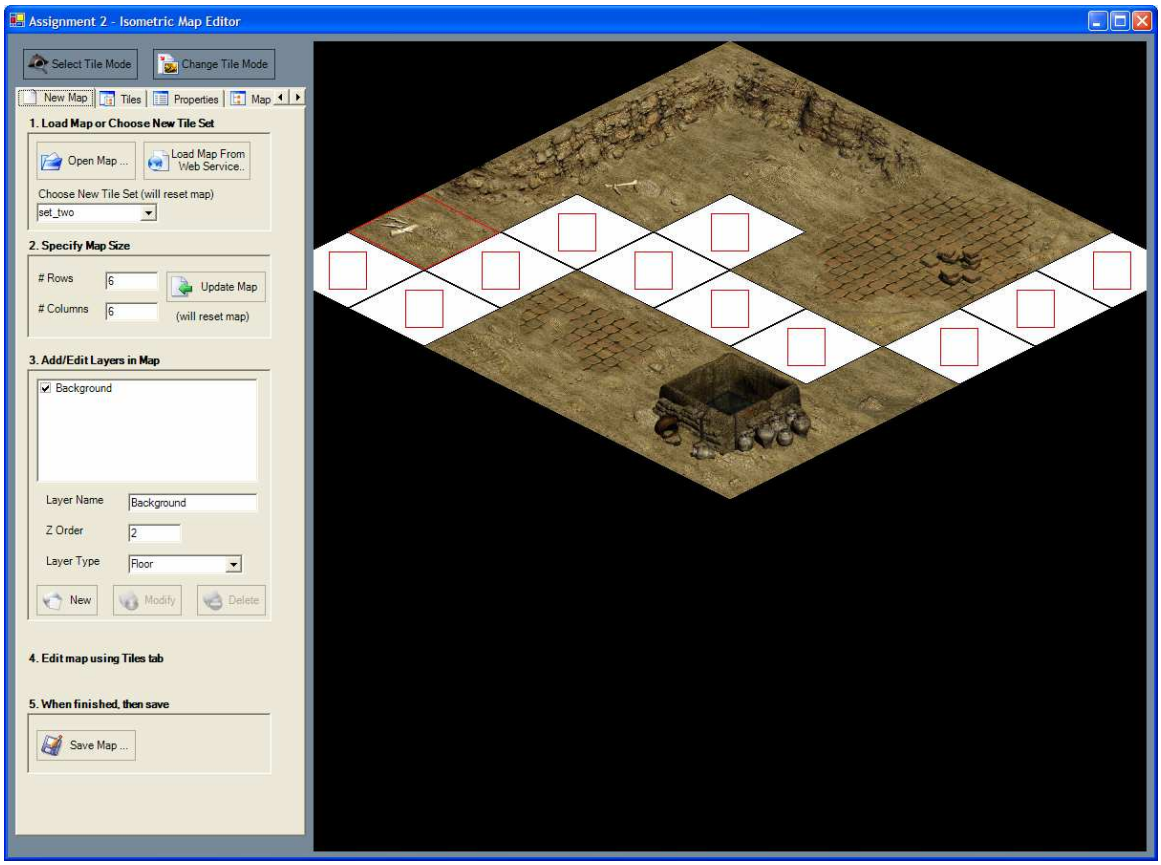


Figure 5

7. VERDICT ON THE GAME AND EDITOR

Students found the graphical nature of the editor enjoyable, and did not find it as difficult as the game. It certainly did expose them to the creation and consumption of web services. However, it did not really capture the typical advantages of service-

oriented computing in general. As a result, the game and editor project is perhaps not the best demonstration wrapper for web services projects.

Nonetheless the semester's experience convinced this author that game development is very much a suitable emerging technology topic. A key part of an emerging technology course as specified in the IS 2002 model curriculum is that the students "implement a project that spans the scope of previous courses" and which exposes students to "contemporary development methodologies" using an emerging environment. This game project accomplished these goals. Using the emerging environment of the .NET Framework along with the more "advanced" technologies necessary for game development (such as XML parsing, 2D and possibly 3D APIs, timers, animation, and threads), students created a project that forced them to integrate and use knowledge and practices from several earlier IS courses. Finally, the game certainly succeeded in capturing the students' interests and efforts (students claimed that they spent considerably more time on this assignment than they usually did for assignments in other courses), and as such this is perhaps justification enough for exposing students to game development in an emerging technology course.

Similarly, web services is also a very suitable topic for an emerging technology course. Current literature in the field strongly suggests that service-oriented computing is the emerging development paradigm of the future. Developing web services using the .NET Framework provides a relatively painless environment for student experiments with this technology.

While the complexity of this game project made it difficult to show off the true benefits of web services, there are many other possible student projects that could demonstrate the power and utility of the service-oriented paradigm (for another take on integrating web services into an upper-level course, see Humphrey, 2004). Some other possible assignment ideas for web services that could be implemented in the confines of a semester are:

A vacation planner. Each student group would become a vacation service provider, such as a hotel, a car rental agency, or an airline. They would then collaborate to decide the standard interfaces for polling information from the service (e.g., availability for specified date, cost for a service) and then implement the web services for their vacation provider.

An enterprise integration application. Each student group would be assigned a particular set of business data – accounting, inventory, sales, customer relations, human resource management, etc. Each group would then collaborate to define interfaces for their services. Each group would then create Windows or Web client application (such as front-to-back sales system) that integrates all this information.

8. CONCLUSION

Despite the difficulties encountered along the way this semester, I feel that both games development and web services can be an important part of an information systems education. The higher-order thinking and programming creativity required for games development can certainly be useful for non-games development. As well, web services are here to stay. Exposing students to this new paradigm will be beneficial for the students' future in the service-oriented computing enterprise of the present and near future.

9. ACKNOWLEDGEMENTS

An earlier version of this paper was presented at the Western Canadian Conference on Computing Education on May 7, 2004 at Okanagan University College, Kelowna, BC.

10. REFERENCES

Becker, Katrin, 2001, "Teaching With Games: The Minesweeper and Asteroids Experience." The Journal of Computing in Small Colleges, Vol. 17, No. 2.

Cerami, Ethan, 2002, Web Services Essentials. Sebastopol, CA: O'Reilly & Associates, Inc.

Channabasavaiah, Kishore, Kerrie Holley, and Edward M. Tuggle Jr., 2004, "Migrating to a service-oriented architecture, Part 1." Available: http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa.

Chatterjee, Sandeep and James Webber, 2003, Developing Enterprise Web Services: An Architect's Guide. New York, Prentice Hall.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, 1995, Design Patterns. Reading, MA, Addison-Wesley.

Giguette, Ray, 2003, "Pre-Games: Games Designed to Introduce CS1 and CS2 Programming Assignments." Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, Vol. 35 No. 1.

Gorgone, J. T., G. B. Davis, J. S. Valacich, H. Heikki, D. L Feinstein, H. E. Longenecker, Jr., 2002, Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, ACM, AIS, and AITP. Available: http://www.is2002.org.

Hashimi, Sayed, 2003, "Service-Oriented Architectures Explained." Available: http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html.

He, Hao, 2003, "What is a service-oriented architecture?" Available: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html.

Humphrey, Marty, 2004, "Web Services as the Foundation for Learning Complex Software System Development." Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, Vol. 36 No. 1.

Jones, Randolph M., 2000, "Design and Implementation of Computer Games: A Capstone Course for Undergraduate Computer Science Education." Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education, Vol. 32 No. 1.

Knorr, Eric , 2003, "Blueprint for Web Services." InfoWorld, Vol. 25, No. 47.

Mungai, Diana, Dianne Jones and Lorna Wong, 2002, "Games to Teach By." Proceedings of the 18th Annual Conference on Distance Teaching and Learning, Madison, Wisconson.

Papazoglou, M. P. and D. Georgakopoulos, 2003, "Service-Oriented Computing." Communications of the ACM Vol. 46, No. 10.

Sondheim, Stephen, 1962, "Comedy Tonight," A Funny Thing Happened On The Way To The Forum. New York, Applause Theatre Books.