Towards a Framework for Migrating Web Applications to Web Services

Asil A. Almonaies Manar H. Alalfi James R. Cordy Thomas R.Dean
School of Computing, Queens University
Kingston, Ontario, Canada
{asil,alalfi,cordy,dean}@cs.queensu.ca

Abstract

Migrating traditional legacy web applications to web services is an important step in the modernization of web-based business systems to more complex inter-business services and interactions. While the problem of migrating various kinds of legacy software systems to a service oriented architecture (SOA) environment has been well studied in the literature, approaches to migrate dynamic web applications to web services are lacking. In this paper we outline the requirements for a semi-automated approach to migrate dynamic legacy web applications to web services-based SOA applications while preserving the original web application's business processes. A manual demonstration of our approach is presented using two examples from Moodle, a popular open source course management system. The migration process is guided by three goals: user interface plasticity, code refactoring and load balancing.

1 Introduction

There are a number of different approaches in the literature to migrate various kinds of legacy software systems to a service oriented architecture (SOA) environment [20, 10, 9, 21, 4]. However, approaches to migrating web applications to web services are lacking [2]. Many legacy web applications are implemented using scripting languages such as PHP [3] or Python [7]. These languages are dynamically typed, reflexive and support dynamic changes to the code. In addition, PHP only fully supported object oriented programming in version 5 [16]. The continuing evolution of these applications has resulted in implementations in mixed programming paradigms. For instance, while some modules in moodle are built using the object oriented paradigm, other modules are non object oriented, while others mix the two paradigms in interesting ways. This mixture of highly dynamic applications mixed paradigms, and the multilingual code bring the challenge to the analysis and refactoring of legacy web application systems for the purpose of migrating them into SOA. To the best of our knowledge this work is the first effort to address these type of applications.

An additional challenges in modernizing a legacy system is the identification of the suitable set of services in the legacy code that have business value. We believe that the need for tools and techniques for analyzing large source code bases to uncover and expose the code that implements business value as services is a crucial area of practical research. These tools and techniques needs to take into consideration the unique characteristics of legacy scripting based web application, and this is one of the purposes of our work.

Web applications can be defined as the set of web pages (.aspx, .jsp, and HTML files), handlers, modules, executable code, and other files, such as images and configuration files, that can be invoked from a web server. Currently, web applications are facing new challenges, such as

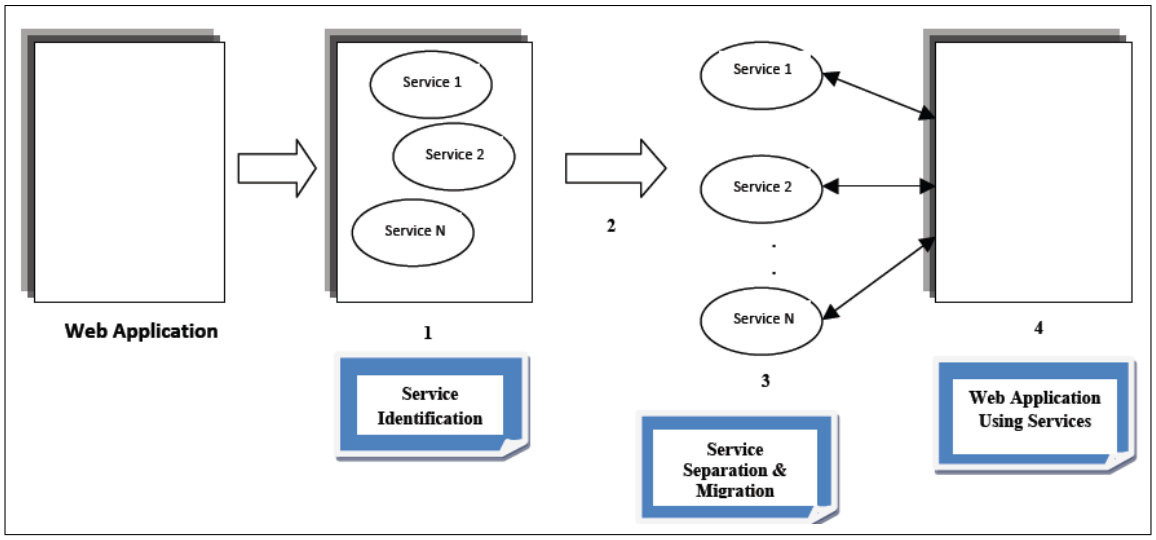

Figure 1: The Proposed Approach

the ability to manage complex processes across multiple users and organizations, to interconnect software provided by different organizations, and the ability to create complex combined business scenarios.

To address these challenges, web services can be used as an enabler technology that provides system-to-system interaction and permits the implementation of business constraints using process control primitives to more easily adapt to these new requirements [6]. The migration of traditional legacy web applications to web services is therefore an important step in the modernization of web-based business systems for the new world of complex inter-business services and interactions.

Using web Services and SOA it is possible to design and build web-based systems and applications using independent, heterogeneous, network-addressable software components. SOA organizes the computer software into a collection of separate services that communicate with each other. It focuses on a loose coupling of the integrated elements, using a set of enabling technologies provided by web services such as XML, Simple Object Access Protocol (SOAP) [25], Web Services Description Language (WSDL) [26] and Universal Description Discover and Integration (UDDI) [14] to achieve flexibility and agility. Using SOA, unnecessary dependencies between systems and software elements are minimized, while maintaining functionality. Service-oriented architecture can help to close the gap between IT professionals and the business professionals in an organization. It also reduces costs and increases return on investment because it focuses on the concept of reusability. A primary reason for its use is to improve business communication, so that the goals of the enterprise can be more directly and readily realized.

The contributions of this paper are:

1. Exploring the issues of the migration of a monolithic web application to SOA, leading to a future semi-automated migration framework. The analysis and refactoring of such system have a unique challenges due to the highly dynamic nature of the legacy system, poorly structured code, multilingual code and weakly typed languages.

2. A manual demonstration on the use of our approach to migrate two of the most interesting functionalities in Moodle, user authentication and the file upload. Our migration process is guided by three goals: user interface plasticity, code refactoring and load balancing.

The remainder of this paper is organized as follows. Section 2 describes our approach for

identifying and separating services. Section 3 describes our first case study and our initial efforts to expose some of the functionalities of the Moodle LMS [13] as web services. Section 4, reviews related work in migrating web applications to web services, and Section 5 summarizes the present state and future directions of our project.

2 Approach

In general, there are three goals when migrating to web services:

- User interface plasticity, which is the capacity of user interfaces to adapt, or to be adapted, to the context of use while preserving usability [5].
- Load balancing, which is a technique to distribute the the workload evenly across two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload [23].
- Code refactoring, which is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behaviour in order to improve some of the nonfunctional attributes of the software [11].

The proposed approach explores the area of moving a dynamic monolithic web application to SOA with significant levels of automation. The result can be considered as a service-oriented web application that implements the endpoints of a web service. The approach can also be used to combine different sets of services from two or more different web applications to construct a new web service application which behaves like the two original applications together.

The proposed approach (Figure 1) consists of four main steps:

1. Identify independent services within the web application. The main challenge in modernizing any legacy system is finding the business services in legacy code by identifying business value in the large code base. The first step in our approach is to identify potential business services in the web application.
2. Separate services in the original web application (that is, break the web application logic into separate services). In this step, and because we are in the first instance migrating PHP-based web applications, we expect to use two technologies:
    - Service Component Architecture (SCA) [19], which provides an easy way to create and access services in PHP. It allows PHP scripts to expose class functions as services and allows class functions to access other services regardless of whether they are local to the current web server or running on a different web server.
    - Service Data Object (SDO) [19], which provides a uniform PHP interface for handling different forms of data and provides a mechanism for tracking changes in data regardless of where the data resides.
3. Migrate the separated services to SOA components. Deciding which services should be selected is a major issue and it is important to have a set of criteria in order to determine the priority for choosing separated services to be included in the target SOA to avoid the selection of redundant or inefficient services.
4. The final step is the re-integration of the selected services into a new service-oriented application. The developed services will be combined to provide the original system functionality as a service-oriented architecture. The result will be an SOA implementation of the original web application functionality.

The approach will be evaluated on an existing production open source monolithic web application to demonstrate that the original application can maintain its original functionality when moved to a web services-based SOA architecture using the approach.

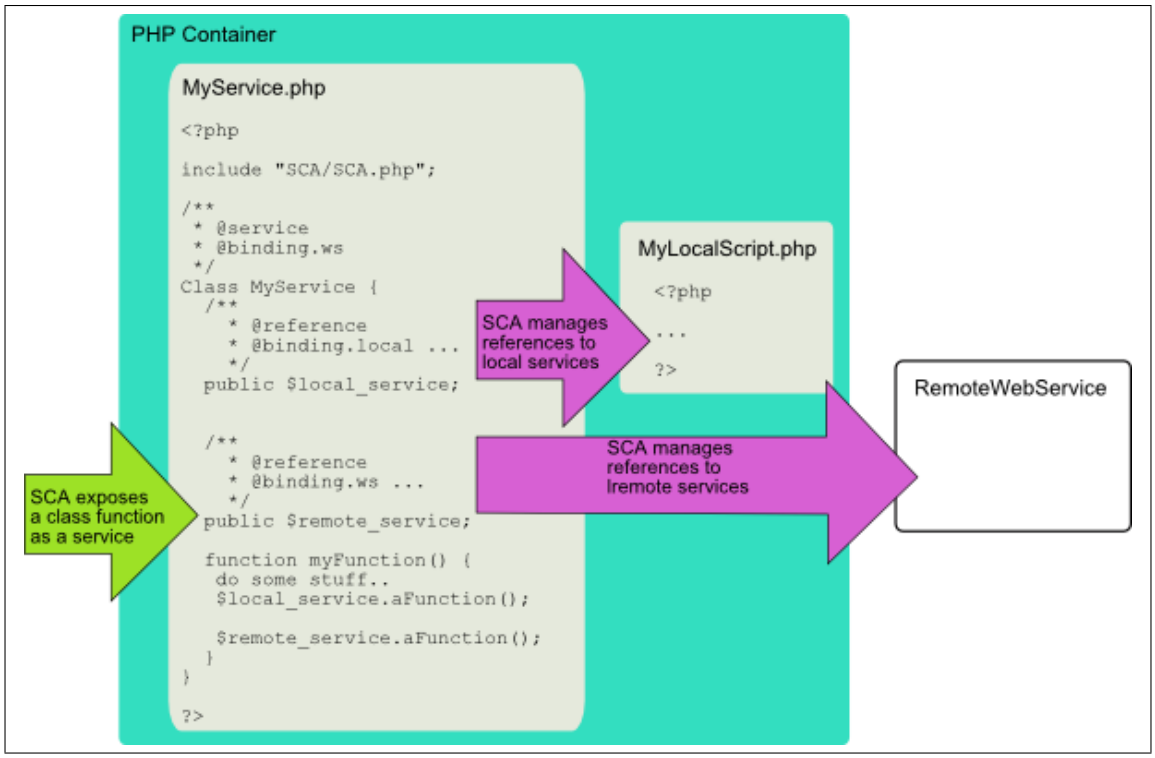

Figure 2: SCA extension (taken from [15])

2.1 PHP SOAP Implementations: SCA

There are several technologies that offer extensions for developing web services in PHP:

- NuSOAP: a group of PHP classes that allow developers to create and consume SOAP web services [18].
- PEAR: SOAP, and a SOAP Client/Server for PHP [24].
- Service Component Architecture (SCA) and Service Data Objects (SDO), which are standards for enabling service-oriented architecture. SCA and SDO are implemented in several programming environments, such as Java, C++, and PHP[12].

In our approach we chose to use SCA and SDO, because they are straight forward to install, adopt and understand. In addition, by using the SCA extension, we are able to implement reusable services using composite references. SCA also provides bindings other than SOAP, such as XML-RPC, JASON-RPC, and REST-RPC [19], making our results more flexible. Figure 2 shows the implementation of SCA in PHP. SCA and SDO consists of two parts. The first part is a dynamic library that extends the php interpreter module of the web server. This extension library scans for annotations given in comments to recognize a PHP class as a web service. This is shown on the left of figure 2. The annotation *@service*, shown at the top of the file `MyService.php` designates the class as a web service. The *@binding* annotation tells the extension which binding to use for the web service. If `ws` is `soap`, then the SOAP bindings is used for the web service. One feature is that the WSDL descriptions are automatically generated when the wsdl parameter is added to the invoking url (e.g. *http* : *//hostname/MyService.php?wsdl*.

The second part of SCA and SDO for PHP is a set of utility functions defined in the include file *"SCA/SCA.php"* used to find and invoke SCA services. The SCA infrastructure handles the invocation of other SCA services,

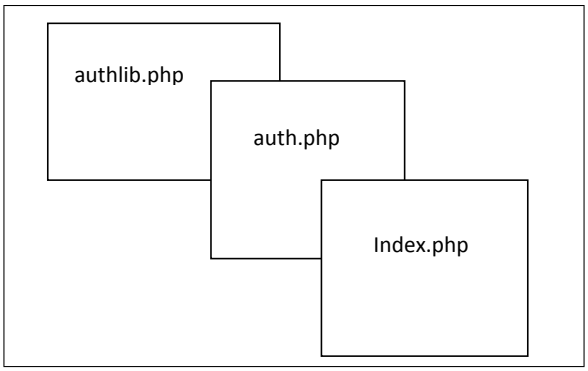

Figure 3: Moodle's login hierarchy

including those written in other implementation languages.

3 Case Study

We use Moodle [13], a popular open source Course Management System (CMS) to explore the issues in SOA migration. Moodle stands for "Modular Object-Oriented Dynamic Learning Environment". It allows teachers to create online courses, which students can access as a virtual classroom. A Moodle home page includes a list of participants (teacher and students), a calendar with a course schedule, and list of assignments. Other interesting features include: online quizzes and forums, where students can post comments and questions, glossaries of terms, and links to other web resources.

Moodle users have four primary roles: administrator, teacher, student, and guest. We chose Moodle because it is widely used internationally, has good documentation and a strong supporting developer community. While Moodle exibits a plug-in architecture, it is accomplished using include files, classes and method calls in a monolithic web application. Since Learning Management Systems (LMS), which provide Internet- based education, are becoming very popular in academia, it is one of the first approaches to provide web services for LMS users using the SCA extension.

We investigate the issues of code refactoring by investigation the migration of Moodle's login functionality to expose it as a service and of user interface plasticity by investigating the file upload service. There are some reusable and reliable functions with valuable business logic embedded in any legacy system. These functions are useful to be exposed as independent services.

While our work is aimed at automating the extraction of the relevant functionality to be selected as services, and in order to develop such a capability we need first to understand how this extraction can be done. In this first experiment, we demonstrate the manual application of our approach to the migration of some of the interesting internal functionalities of Moodle, such as the login process, that can be reused. We have manually analyzed Moodle in a top-down manner, dividing basic functionalities into sub-functionalities in order to gain insight into the application, and as a result we have identified a number of potential independent services.

3.1 Example 1: Login Process

Moodle provides a single login page to all its users (administrator, teacher, student and guest), where a user name and password are required (administrator, teacher, student) and their information validated before the user is allowed to continue. The code that generates and processes page is in *login/index.php*, relative to the Moodle base directory. Login is thus the first functionality we have identified as a potential service.

```php
<?php
    require_once("../config.php");
    include('authenticate.php');

    if ($_SERVER['REQUEST_METHOD'] == 'GET'){
      // return form
      include('login_form.htm');
    } else {
      $user = $_POST['username'];
      $pwrd = $_POST['password'];
      if (login($user,$pwrd)){
        $_SESSION['username'] = $user;
        include('login_success.htm');
      } else {
        include('login_fail.htm');
      }
    }

?>
```

Figure 4: Simplified login/index.php page

```php
<?php
 require_once("../config.php");
 include("auth_manual.php");

 function login($u,$p){
    // in real Moodle, this selects the
    // authentication method
    // from the configuration file.

      $mauth = new auth_manual ();
      return($mauth -> login($u,$p));
   }
?>
```

Figure 5: Authentication Code

3.1.1 Moodle's Login Identification

Moodle's login functionality is already structured as multiple plugins which are provided using an factory routine that chooses the correct login method based on configuration information. While one approach provides a new plug-in for an SOA method, we instead choose to investigate migrating one of the existing plug-ins to an SOA environment. We start analyzing the code manually to identify the candidate functionalities. We have identified several of the authentication plugins used in Moodle:

- Manual authentication : accounts are created manually by an administrator.
- Email authentication: accounts are created manually by the users.
- LDAP authentication: accounts information are on external Lightweight Directory Access Protocol server.
- Nologin authentication: accounts are suspended.

We choose to investigate the first option, manual authentication which does a simple check against the Moodle database. This functionality is spread over the following php pages:

- *login/index.php* : which is the default login page.
- *lib/moodlelib.php*: contains the function *authenticate_user_login()* called by index.php which get the list of all the enabled authentication plugins.
- *lib/auth/manual/auth.php*: contains the manual authentication.
- *lib/authlib.php*: contains all the authentication plugins types.

```
<?php
require_once("../config.php");

class auth_manual {
  function login($username,$password){
    $p = md5($password);

    $query="SELECT * FROM mdl_user where
        username='$username' and password='$p'";
    $result=mysql_query($query);
    echo mysql_error();
    $num=mysql_num_rows($result);

    if($num==0) { // if result->password == $p
      return false;
    } else {
      return true;
    }
  }
}

?>
```

Figure 6: Manual authentication module

The Moodle login logic hierarchy is as shown in (Figure 3). From the above authentication methods supported by Moodle, we will adapt the manual authentication method, which just does a simple check of the user name and password hash against the Moodle database.

3.1.2 Moodle's Login Refactoring

We begin with the Moodle login page in *login/index.php* which is responsible for displaying the form that allows the Moodle users to enter their information. It also includes the authentication code that uses the function login which takes two parameters; the username and password. We have produced a simplified version of the login code to illustrate the differences shown in figure 4.

Figure 5 shows the simplified code in authenticate.php. In Moodle, this is an factory method that instantiates the class that implements the login method given in the configuration file and invokes the *login* method. We have simplified to code to instantiate and call the class that implements the manual method of user authentication.

Figure 6 shows our simplified version of the original manual authentication code. While simplified it still reflects the basic functionality of Moodle's manual method. The user password is hashed using md5 and the values are checked in the internal Moodle database. If there, the function returns true, otherwise false.

3.1.3 Moodle's Login New Service

In this case the migration to SOA is straightforward. We create a copy of the original code with a new name (file and class name must match in php). We add the include statement for the SCA library, and annotations for the class and method. This includes the *@service* and *@binding.soap* annotations for the class, and the parameter and result annotations for the login method. This enables the class as a remote service.

The original class is replaced by a wrapper as shown in figure in (Figure 8). The PHP version of SCA automatically generates a WSDL file if the php file is invoked remotely with the *wsdl* parameter. Thus the wrapper starts by generating a WSDL file from the remote service and saving it locally (production code cache the WSDL file). This file is then used to instantiate the service and then call the remote service. The result of the service is returned to the calling function (i.e. *login*).

Thus authenticating a user from *login/index.php* is accomplished by calling the factory *login* routine in authenticate.php. The factory routine instantiates the wrapper class in authmanualSOA.php and invokes the *login*, which in turn calls the remote service.

```
<?php
   require_once("../config.php");
   include 'SCA/SCA.php';
   /**
   *
   * @service
   *
   * @binding.soap
   *
   */
   class auth_manualSOA1remote {
   /**
   *@param string $username
   *
   *@param string $password
   *
   *@return boolean
   */
   function login($username,$password){
   $p = md5($password);
   $query=" SELECT * FROM mdl_user where username='$username' and password='$p'";
   .
   .
}
?>
```

Figure 7: Remote Version of Manual Authentication

```
<?php
   require_once("../config.php");
   include 'SCA/SCA.php';
   class auth_manualSOA1 {
   function login($u,$p){
   $f = file_get_contents('http://pices.ee.queensu.ca/moodle/login/auth_manualSOA1remote.php?wsdl');
   file_put_contents('auth_manualSOA1remote.wsdl',$f);
   $service = SCA::getService('./auth_manualSOA1remote.wsdl');
   $result = $service->login($u,$p);
   return $result;
   }
}?>
```

Figure 8: Wrapper Class to Call Remote Service

3.2 Example 2: File Handling

As a course management system, Moodle provides a facility to allow users with the teacher role (i.e. professors, TA's) to provide files for students to download. The file functionality in moodle allows teachers to upload, organize, rename, delete and generate zip archives of the files.

In this experiment we focus on converting the file uploading functionality to a service. Once converted to a service, the Moodle's multi-page based file management interface can be converted to a more interactive Ajax interface. Thus the objective of this activity is user interface plasticity. As with the login functionality, the base logic of the file management functionality is located in *files/index.php* in the Moodle directory.

3.2.1 Moodle's File Uploading Identification

The main code of the file module is located in *files/index.php*. There are also several library files used to implement file uploading, *adminlib.php* (which contains functions that is used by the administrators), *filelib.php* and *file.php*(which contain functions for analyzing file content and file types), and *uploadlib.php* which contains the class that manages the upload functionality. What makes this more difficult is that presentation logic and file management is spread throughout the upload class.

```
<?php
  class upload_manager {
  var $files;
  var $config;
                . . . .

   function upload_manager($inputname='', $deleteothers=false,$handlecollisions=false   $course=null,
   $recoverifmultiple=false, $modbytes=0, $silent=false, $allownull=false, $allownullmultiple=true) {
                .
                .
     }

   function validate_file(&$file) {
     if (empty($file)) {
     return false;
   }
   if (!is_uploaded_file($file['tmp_name']) || $file['size'] == 0) {
                . . . .
   /** Moves all the files to the destination directory */
   function save_files($destination) {
                . . . .
   if (move_uploaded_file($this->files[$i]['tmp_name'], $destination.'/'.$this->files[$i]['name'])) {
     chmod($destination .'/'. $this->files[$i]['name'], $CFG->directorypermissions);
     $this->files[$i]['fullpath'] = $destination.'/'.$this->files[$i]['name'];
     $this->files[$i]['uploadlog'] .= "\n".get_string('uploadedfile');
     $this->files[$i]['saved'] = true;
     $exceptions[] = $this->files[$i]['name'];
     .
     .
?>
```

Figure 9: The Moodle upload manager

```
<?php
    require 'SCA/SCA.php';
    /**
    * An SCA Component that upload a file
    *
    * @service
    */
    class UploadService
    {
    var $tmp_name;
    var $filename;
    var $directory;
    function movefile() {
    $tmp_name = $_FILES['file']['tmp_name']; ;
    $filename = $_FILES['file']['name'];
    $directory = "upload/";
    if (is_uploaded_file($tmp_name)) {
      if (move_uploaded_file($tmp_name,$directory . $filename)) {
      return "File Uploaded Successfully";
    } else return "File Uploading Failed";
    }
    } else {
    return "false";
  . . .
?>
```

Figure 10: The uploadservice.php

For example the upload manager generates HTML directly to the user when it encounters errors, either in the format or content of the files, or in file management.

The method *move_uploaded_ file(string $filename,string$directory)* of *uploadlib.php* is used to check the validity of the file (file type, absence of malware, etc.) which is uploaded via an HTML form. If it is valid then it is moved to the destination directory (Figure 9).

3.2.2 Moodle's File Uploading Refactoring and New Service

The file management routines from the library are collected into a single class and converted to a local service. Figure 10 shows the *movefile* method of the class. The upload manager class is then converted to call the SOA functions. The collection of the file management code into a service separates the management of the files from the generation of the error messages.

4 Previous Work

Service-oriented architecture (SOA) migration is an architectural migration from any non-SOA system to a system that follows the service-oriented architecture principles, in order to achieve a new maintainable service-oriented architecture implementation of the system. The major benefits of adopting service oriented architecture as a design framework is the ability to realize rapid and low-cost system development, to improve overall system quality, and to better enable integration with other systems.

Several methods in the literature studied the problem of migrating traditional legacy system to web services. We discuss some of them here, and for a comprehensive discussion of other approaches, interested user can refer to our previous work [2], in which we present a comparative study of modernization approaches for leveraging /exporting legacy systems to SOA.

Aversano et al. [4] present a case study in which a COBOL system is migrated to web-based service oriented architecture. The legacy system is divided into user interface and server (application logic and database). The user interface has been migrated into a web browser shell using Microsoft Active Server Pages and the VBScript scripting language, and the MORPH approach has been used to map the components of the existing interface into the new web based interface. While the server has been wrapped and integrated into the new web-enabled system with dynamic load libraries written in Microfocus Object COBOL, loaded into Microsoft Internet Information Server (IIS), and accessed by the ASP pages.

H. M. Sneed and S. H. Sneed [21] discuss a tool-supported method, where the legacy code is a COBOL program wrapped behind an XML shell allowing individual functions within the programs to be offered as web services to any external user.

Smith [20] and Lewis et al. [10, 9], discuss a migration technique called the Service-Oriented Migration and Reuse Technique (SMART). It is a technique that helps organizations analyze legacy systems to decide whether their functionality can be reasonably exposed as services in a service-oriented architecture.

Little work has been done to migrate web applications to SOA. A sample of such efforts is the work done by Tatsubori and Takash [22]. The authors present a framework named H2W, which can be used for constructing web service wrappers for existing, multi-paged web applications. H2Ws contribution is mainly in its service extraction step. The authors propose a page-transition-based decomposition model and a page access abstraction model with context propagation. With the proposed decomposition and abstraction, developers can flexibly compose a web Service wrapper of their intent by describing a simple workflow program. In [8] the paper discusses how web services can be used to leverage web applications in a similar way in general.

Vijaya and Rajan [17] focus on exploring web services features, how they could effectively shape the e-learning process, and advantages and compromises for the migration from traditional distributed application development to web services enabling technology. Mainly they explore the benefits of converting to web services, without introducing any specific approach to conversion. Most of the work done in this area is similarly general, where the focus is on discussing the benefits of using web services to leverage web application, without introducing a full framework for addressing the problem.

The work done by Ajlan and Zedan [1] is to introduce web services to Moodle. The focus of the paper is to expose the assignment module of Moodle as web services. UML diagram (collaboration diagram) is used to analyze it and to capture the necessary information needed to

expose the assignment module as web services. Furthermore, NuSOAP is used to create and consume web services. The goal of our paper is to introduce an approach for moving legacy web application to web services in a service-oriented architecture.

5 Conclusion and Future Work

The proposed approach represents a new approach to the problem of migrating legacy systems to service-oriented architecture. It is one of the first approaches to explore the area of moving a monolithic web application to SOA, and the first with a complete approach and significant levels of automation. We presented a simplified version of the Moodle login as a service, which we were able to successfully authenticate users into Moodle by checking the username and password provided against the Moodle database.

As future work, we will expose the file uploading functionality via SOAP protocol . Also we will choose more functionality to be exposed as services within Moodle to achieve the user interface plasticity and load balancing goals respectively. We analyzed both functionalities in order to identify the potential service within each function, which is done manually at this stage. We are also going to automate the service identification process by applying TXL.

In addition, as a demonstration of the flexibility of the results of the approach, we will combine the services extracted from two different web applications. This will demonstrate a new hybrid system that uses mixed extracted services from the applications to offer new higher-level business functionalities that use the services of both.

Acknowledgements

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada and by an IBM Center for Advanced Studies Faculty Fellowship.

About the Authors

Asil Almonaies is a PhD candidate in the School of Computing at Queen's University working under the supervision of Profs. James Cordy and Thomas Dean. She received her M.Sc. and B.Sc in Computer Engineering from Kuwait University. Her research interests are in the area of software engineering, focusing on web applications, web services, service-oriented architecture, and legacy systems migration.

Manar Alalfi is a postdoctoral fellow in the Software Technology Laboratory of the School of Computing at Queens University, working with Prof. James Cordy. Dr. Alafi received her Ph.D. from Queens University in 2010. Her major specialization is software engineering, focusing on security analysis in web applications, studying types of web application vulnerabilities and proposing and developing techniques to harden software systems to survive malicious attacks. Her other research interests include service oriented architecture and model driven engineering for automotive systems.

James Cordy is Professor and past Director of the School of Computing at Queens University. From 1995-2000 he was vice-president and chief research scientist at Legasys Corporation, a software technology company specializing in legacy software system analysis and renovation. As leader of the TXL source transformation project he is the author of more than 130 refereed contributions in programming languages, software engineering and artificial intelligence. Dr. Cordy is an ACM Distinguished Scientist, a senior member of the IEEE, and an IBM CAS visiting scientist and faculty fellow.

Thomas Dean is an Associate Professor in the Department of Electrical and Computer Engineering at Queens University and an Adjunct Associate Professor at the Royal Military College in Kingston. His background includes research in air traffic control systems, language formalization, and five and a half years as a Sr. Research Scientist at Legasys Corporation where he worked on advanced software transformation and evolution techniques in an industrial setting. His current research interests are software transformation, web site evolution and network application security.

References

[1] A. Ajlan and H. Zedan. E-learning (MOODLE) Based on Service Oriented Architecture. In *the EADTU's 20th Anniversary Conference, Lisbon, Portugal, 8-9 November, Lisbon-Portuga*, pages 62–700, 2007.

[2] A. Almonaies, J.R. Cordy and T.R. Dean. Legacy System Evolution towards Service-Oriented Architecture. In *International Workshop on SOA Migration and Evolution*, pages 53–62, 2010.

[3] Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Loopes, Hannes Magnusson, Georg Richter, Damien Seguy, and Jakub Vrana. PHP Manual. `http://www.php.net/manual/en/index.php`, last accessed Aug 2011.

[4] L. Aversano, G. Canfora, A. Cimitile, and A. De Lucia. Migrating legacy systems to the web: an experience report. In *Proceedings of Fifth European Conference on Software Maintenance and Reengineering*, pages 148–157, 2001.

[5] Joëlle Coutaz. User interface plasticity: Model driven engineering to the limit! In *2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–8, 2010.

[6] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1:113–137, 2002.

[7] G. Van Rossum. Python programming language. `http://www.python.org/`, last accessed Aug 2011.

[8] K. Dezhgosha and S. Angara. Web services for designing small-scale Web applications. In *the 2005 IEEE International Conference on Electro Information Technology*, 4 pages, 2005.

[9] G. Lewis, E. Morris, L OBrien, D. Smith, and L. Wrage. Smart: The service-oriented migration and reuse technique. In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 222–229, 2005.

[10] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In *In Proceedings of the Conference on Software Maintenance and Reengineering*, pages 15–23, 2006.

[11] Martin Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[12] Caroline Maynard, Graham Charters, Matthew Peters, and Simon Laws. SCA/SDO for PHP. `http://pecl.php.net/package/SCA_SDO`, last accessed Aug 2011.

[13] Moodle Trust. Moodle. `http://Moodle.org`, last accessed October 2010.

[14] OASIS. UDDI Version 2.03 Data Structure Reference. `http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm`, last accessed September 2010.

[15] Open SOA Collaboration. SOA PHP Homepage. `ihttp://www.osoa.org/display/PHP/SCA+with+PHP`, last accessed Aug 2011.

[16] Php.net. PHP 5 ChangeLog. `http://www.php.net/ChangeLog-5.php`, last accessed Aug 2011.

[17] Amala Vijaya Selvi Rajan and Jim Otieno. Leveraging traditional distributed applications to web services for e-learning applications. In *15th International Workshop on Database and Expert Systems Applications*, pages 430–435, 2004.

[18] Scott Nichol. Introduction to NuSOAP. `http://www.scottnichol.com/nusoapintro.htm`, last accessed September 2010.

[19] Simon Laws. SCA with PHP. `http://www.osoa.org/display/PHP/SCA+with+PHP`, last accessed September 2010.

[20] D. Smith. Migration of legacy assets to service-oriented architecture environments. In *Proceedings of the 29th International Conference on Software Engineering*, pages 174–175, 2007.

[21] H. M. Sneed and S. H. Sneed. Creating web services from legacy host programs. In *Proceedings of the Fifth IEEE International Workshop on Web Site Evolution*, pages 59–65, 2003.

[22] Michiaki Tatsubori and Kenichi Takashi. Decomposition and abstraction of web applications for web service extraction and composition. In *IEEE International Conference on Web Services*, pages 859–868, 2006.

[23] TechGuruLive. What is a hardware load balancer? `http://techgurulive.com/2011/01/17/what-is-a-hardware-load-balancer/`, last accessed October 2010.

[24] The PHP Group. Package Information: SOAP. `http://pear.php.net/package/SOAP/redirected`, last accessed September 2010.

[25] W3C. Simple Object Access Protocol (SOAP) Version 1.2. `http://www.w3.org/TR/soap12`, last accessed September 2010.

[26] W3C. Web Services Description Language (WSDL) Version 1.1. `http://www.w3.org/TR/wsdl11`, last accessed September 2010.