A Framework for Migrating Web Applications to Web Services

Asil A. Almonaies, Manar H. Alalfi, James R. Cordy, and Thomas R. Dean

School of Computing, Queens University
Kingston, Ontario, Canada
{asil,alalfi,cordy,dean}@cs.queensu.ca

Abstract. In this paper, we present a framework for semi-automatically migrating monolithic legacy web applications to service oriented architecture (SOA) by separating potentially reusable features as web services. Software design recovery and source transformation techniques are used to automatically analyze and reprogram web application code to migrate existing web-based systems to support inter-business services and interactions. Such modernization helps make web applications more flexible, allowing them to more easily integrate functionality with other systems and respond to rapidly changing business needs. While the problem of migrating other kinds of legacy software systems to an SOA environment has been well studied in the literature, approaches to migrating legacy web applications to web services are lacking. We demonstrate our framework on the analysis and automated restructuring of an existing PHP web application, by migrating integrated internal features to independent, reusable web services.

1 Introduction

Service Oriented Architecture (SOA) is an increasingly important software architecture, designed to flexibly interconnect software components in response to rapid changes in the business environment. In SOA, applications are split into separate software services that can be maintained independently and easily reused. In order to provide the advantages of SOA in the context of the world wide web, Web Services are used as an enabling technology, allowing web-based business functionalities to interconnect in an object-model-neutral manner.

At present the vast majority of production web applications use a monolithic standalone software style. These applications are designed largely without clear modularity, which makes their maintenance and enhancement in response to rapidly changing business requirements a difficult task. Rather than re-implement the business functionality of these applications as services from scratch for the new world of interoperation and reuse, web providers would prefer to preserve their investment by migrating their existing web application functionality to web services. These dynamic legacy web applications are simply too important to be discarded, and thus they must be reused.

Several modernization approaches to move legacy systems to SOA environments have been described in the literature. However, to our knowledge only a few research studies have attempted to address the problem of automatically moving monolithic legacy dynamic web applications to SOA. Moreover, the work that is done in this area [1, 2, 3] is very general, discussing the benefits leveraging existing web applications in

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 384–399, 2013.

moving to web services, without proposing any practical framework for actually implementing the change.

The nature of monolithic dynamic web applications, often with mixed paradigms and multi-lingual code, makes analysis and refactoring for the purpose of migration to SOA a challenging and error-prone problem. Automation of the migration process reduces human intervention, which reduces the time and cost and increases the consistency of the migrated code.

In this paper we present a framework and tool set that largely automates the migration from monolithic PHP web applications to SOA. Software design recovery and source transformation techniques are used to assist in automating the migration from legacy web applications to service-oriented web services. The result can be considered to be a service-oriented web application that implements the endpoints of a Web Service. The framework 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 contributions of this paper are:

- A framework using an iterative process of incremental steps to analyze and reprogram existing web applications to web services based on the Service Component Architecture (SCA) web services standard.
- An automatic extraction process to extract and separate identified business features in dynamically-typed scripting languages as object-oriented classes.
- An automatic process for inferring the types of parameter values in dynamically–typed scripting languages using instrumentation and coverage testing.
- An automatic process for converting an object-oriented class into an SCA service component.
- A prototype set of tools using source analysis and transformation to automate the reprogramming of web applications written in PHP to extract and separate identified business features as web services.
- A demonstration of the framework and tool set in extracting web services from SCARF, a monolithic web application for a conference and research paper discussion forum, and automatically reprogramming it to use these services.

The rest of this paper is organized as follows: Section 2 gives an overview of our framework for SOA migration, and Section 3 details the steps of our automated iterative migration process. Section 4 presents a case study using our framework and automated process in practice. Section 5 relates our work to other work in SOA migration, and finally Section 6 summarizes our conclusions and outlines our future work.

2 A Web Application to SOA Migration Framework

Our proposed framework uses a new approach to the problem of legacy system migration to service-oriented architecture. It is one of the first approaches to explore the area of moving a monolithic web application to SOA with significant levels of automation. The proposed framework (Figure 1) consists of two main steps, Service Identification and Service Migration, to produce a new application using web services.

2.1 Service Identification

While our work concentrates on the service migration aspect of the problem, one of the main challenges in modernizing a web legacy system is the identification of potential service functionality that may have business value. Our approach does not attempt to solve the identification of services, rather we leverage the results of other research such as the work done by Asuncion et al.[4], which uses goal-based, model-driven and service-oriented approaches to identify business rules in the application.

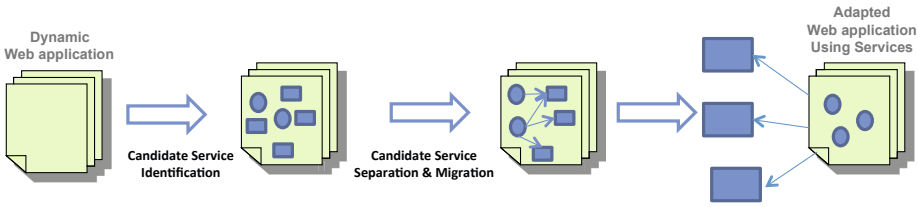

Fig. 1. A Migration Framework

In our framework, the output of the service identification step is a marked-up version of the web application source code in which sections of code with the desired business functionality have been identified as the operations of a candidate service. This tagged candidate service is then the input to our automated migration process.

In this paper we carried out the identification process manually. Based on the functionalities that we wanted to extract from the adapted web application, we identified each potential operation of the candidate service using XML markup of the application source. In our identification notation, each candidate service operation is marked up using a <*service function* = *function-name*> tag, where *function-name* is a user-suggested name for the candidate service operation (Figure 2).

2.2 Service Separation and Migration

Candidate service migration is the process of separating each identified candidate service into a separate class , extracting it from the original application code, converting the created class into a separate independent service, and adapting the original application code to use the separated service. Once extracted and migrated to a separate service, the extracted service is used by the adapted original application as a client, and can also be easily used by other web applications.

Legacy web applications are generally implemented using scripting languages such as PHP [5] or Python [6]. These languages are dynamically typed, reflexive and support dynamic changes to the code. The nature of monolithic dynamic web applications, often with mixed programming paradigms, makes the analysis and refactoring of web application source code challenging. Thus the process of separation & migration of candidate services is time consuming, technically complex and error-prone.

While there are a number of different approaches to migrating various kinds of legacy software systems to a service oriented architecture in the literature [7, 8, 9], approaches

```
<?php
include ("welcome.html");

$fname = "John";

<service function = Reverse>
     $name = strrev ($fname);
</service>

echo "My name is".$name;
?>
```

Fig. 2. Example Marked Candidate Service Operation

to migrating web applications to web services are lacking [10]. This lack of other approaches, and the clear need for automation to assist in web application migration is the focus of the work of this paper. The concrete goal of our research is to to automate the separation and migration of identified candidate services in PHP-based web applications to web services using IBM's Service Component Architecture (SCA) standard. While our work concentrates on PHP in this paper, the same process and strategy can be easily adapted to other web application languages and technologies.

3 Automating Service Migration

Our process for automating service separation and migration consists of several steps, each implemented using a TXL [11] source transformation of the PHP web application code. The five steps of our process are (Figure 3) :

1. Candidate Service Refactoring
2. Candidate Service Separation
3. Parameter Type Inference
4. Service Component Conversion
5. Database Refactoring

The following sections describe each of these steps in detail.

3.1 Candidate Service Refactoring

The input for the first step is the marked up source code of the web application which identifies PHP code sections as potential operations of the candidate service. In the simple example of Figure 2, a PHP code section is marked as the candidate service operation "Reverse".

The refactoring step automatically creates a PHP function for each of the marked up candidate code sections, and wraps them in a new class for the candidate service. Parameters and results of the functions are inferred from the dependencies of the code sections on their context, and the original code sections are replaced by parameterized calls to the functions of the new candidate service class.

When this step is complete, the application has been refactored to separate the original marked code sections into functions of the separate class (Figure 4). The user provides a name for the new class, in this case "Example".

3.2 Candidate Service Separation

In the next step, we automatically separate the new candidate service class into a separate PHP class file and generate the appropriate PHP code necessary for the original program to use it, including include directives for the separated class file and creation of an instance object for use in the original code.

As part of the separation, we create a constructor class for each of the operations wrapped in class, called the *return class*, which acts as a dictionary to contain the returned values of the operation. The results of the candidate service separation step on our simple example candidate class are shown in Figure 5.

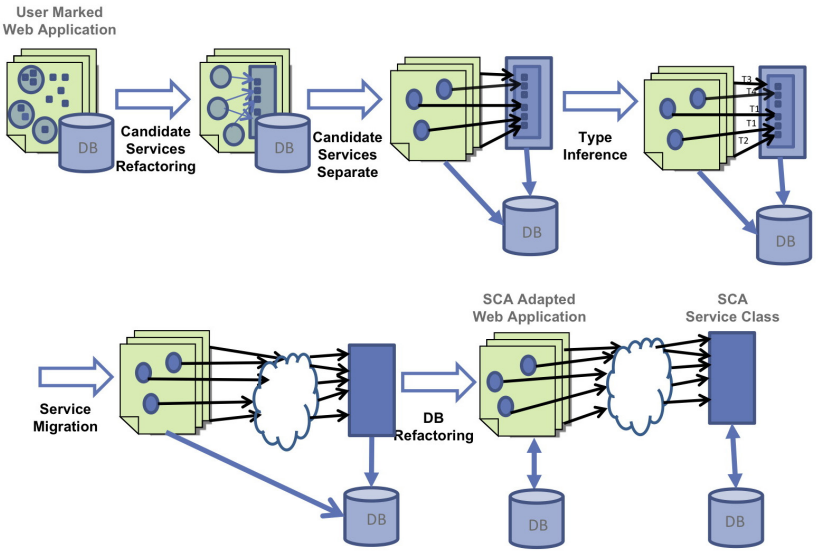

Fig. 3. Steps of our Automated Process for Service Migration

```
class Example {
   function Reverse ($name, $fname) {
      $name = strrev ($fname);
      return new Reverse_return ($name);
   }
}
```

Fig. 4. Example Class Generated by the Refactoring Step

3.3 Parameter Type Inference

Like most web application languages, PHP is a dynamically typed language, and types of function parameters are not normally specified. A parameter simply has whatever type it takes on at run time. Parameters to service operations, by contrast, must be specified as part of the service description.

Thus in this step we first instrument each function of the refactored and separated candidate service class to dynamically capture parameter types, and then run the instrumented application to cover execution of every candidate service operation function. The instrumentation stores in a file a table of each function annotated with the types of the parameters it receives when actually run. In some cases, parameters end up with a NULL type, if the corresponding variable has not been set when the function is called. In this case we delete the NULL values as they do not affect the output.

The type table file is then used to explicitly annotate the parameters of the service operation functions of the candidate service class with their expected types. These parameter types are required in the Service Component Conversion step (Section 3.4) both for creating the Web Services Description Language (WSDL) service description of the new service, and for creating SCA parameter annotations for the operations of the new service. The result of the parameter type inference step is a fully typed version of the separated candidate class file (Figure 6).

```php
<?php
include ("welcome.html");

include_once "Example_return.php";
include_once "Example.php";
$Example_obj = new Example ();

$fname = "John";

$Reverse_return_obj = $Example_obj -> Reverse ($name, $fname);
$name = $Reverse_return_obj -> name;

echo "My name is".$name;
?>
```

(a) Refactored Original Code after Candidate Service Class Separation

```php
<?php

include_once "Example_return.php";

class Example {
   function Reverse ($name, $fname) {
      $name = strrev ($fname);
      return new Reverse_return ($name);
   }
}
?>
```

(b) Separated Candidate Service Class

```php
<?php
class Reverse_return {
   public $name;
   public function __construct ($name) {
      $this -> name = $name;
   }
}
?>
```

(c) Return Value Constructor Class for Reverse Operation of Separated Candidate Service Class

Fig. 5. Example Refactored and Separated Candidate Service Class

```php
<?php
include_once "Example_return.php";

class Example {
   function Reverse (NULL $name, string $fname) {
      $name = strrev ($fname);
      return new Reverse_return($name);
   }
}
?>
```

Fig. 6. Example Refactored and Separated Service Class after Type Inference

3.4 Service Component Conversion

After inferring parameter types of the separated candidate service class operation functions, we are ready to reprogram the class into a real service component. In this step we convert the separated candidate service class file into an SCA service component, by adding the required SCA annotations to the class and each of its operation functions specifying the name, number and types of the expected service operation message parameters. As part of this conversion, the Web Service Description Language (WSDL) service description file is created automatically by the SCA technology.

```php
<?php
include ("welcome.html");

include_once "Example_return.php";
include_once ("SCA/SCA.php");
$Example_obj = SCA :: getService ("Example.wsdl");

$fname = "John";

$Reverse_return_objStr = $Example_obj -> Reverse ($name, $fname);
$Reverse_return_obj = unserialize ($Reverse_return_objStr);
$name = $Reverse_return_obj -> name;

echo "My name is".$name;
?>
```

(a) Converted Original Application as SCA Client

```php
<?php
include_once "Example_return.php";
include "SCA/SCA.php";

/**
   * @service
   * @binding.soap
   */
class Example {
   /**
      * @param string $fname
      * @return string
      */
   function Reverse (string $fname) {
      $name = strrev ($fname);
      return serialize (new Reverse_return ($name));
   }
}
?>
```

(b) Converted Candidate Service Class as SCA Service

Fig. 7. Example Converted to a Web Service-based Application

In order to create an SCA component several steps are required. SCA service type annotations must be added to each of the service operation functions of the candidate service class to specify the types of parameters and return values of the operation. The SCA interface and SCA service annotations must be generated for the candidate service class to specify the service and its service binding (in the case of our conversions, the SOAP messaging protocol). And finally, the original adapted web application must be converted to a service client of the WSDL service description and SCA protocol.

Figure 7 shows the result of applying these transformations to the candidate service class file and refactored original application to create an SCA-based client/server relationship using the new web service.

3.5 Database Refactoring

In the final step of our migration, the original application database is refactored to separate those tables used only by the new separated service into a separate database, and remove them from the original application database. This allows the new web service to be used by other applications independently of the original. In our current implementation of the framework, this final step is done manually when required.

4 A Case Study: SCARF

In the previous sections we have outlined our framework for automatic migration of web applications to SOA using a sequence of source transformations that take identified potential service operations in the application code to separate reusable SCA web services. Our running example has demonstrated the application of the process to a small but representative toy web application.

Thus far we have used our framework on two real web applications, the Moodle course management system [12], a large production web application used by thousands of students and instructors worldwide, and SCARF, the Stanford conference and research forum, a research discussion forum application [13]. Due to space limitations, in this paper we only show the use of our framework in separating and migrating the paper management functionality of SCARF to a web service.

4.1 The SCARF Paper Management Subsystem

SCARF [13] is a PHP-based web application designed to help researchers and conference administrators create and maintain discussion forums for their research papers. In SCARF, papers are uploaded and stored in a database where users can view, comment and edit them, as well as organize them into sessions. SCARF is intended to support interactive conferences such as SIGCOMM, for which it was originally developed.

4.2 Step 1: Paper Management Service Identification

Our plan is to identify and separate a new web service for the research paper management aspects of SCARF, separating it from the user interface code of the web application so that it can be accessed and reused by other applications. The paper management

system in SCARF supports several operations. For example, users can download a specific paper, edit the content of a paper, and add a new paper to the forum.

We begin by analyzing the SCARF source to identify the functionalities related to paper management. The business logic of the paper management functionality is spread over five PHP pages:

- *editpaper.php*: Logic to enable an authenticated user to add a new paper to the forum or edit the information of existing papers.
- *showpaper.php*: Logic to access specific paper details, such as name, authors, abstract, comments, the paper document and auxiliary files.
- *showsession.php*: Logic to show all papers available in a specific session with information about them.
- *getpaper.php*: Logic to download a paper.
- *getfile.php*: Logic to download an auxiliary file associated with a paper.

Each of these pages contains sections of code that provide particular discrete operations that we can identify as part of our candidate paper management service class, interspersed with user interface code to present and interact with the page. Figure 8 shows the tagged candidate service operation code sections for the paper management functionality of SCARF in the *editpaper.php* page.

4.3 Step 2: Refactoring

While candidate service operations are often contained in a single PHP source file, in the case of the SCARF paper management functionality, the code is spread over several different PHP source of the application. To handle this we use our refactoring transformation to generate several candidate service classes for the operations, one from each PHP page, and merge the results into a single unified candidate service class (Figure 9) before conversion to an SCA service.

We run our refactoring transformation in turn on each of the five tagged source pages, generating a new separate candidate service class for each one, while adapting the original page to use the new service. By specifying to the refactoring process that the new candidate service classes should each have the same name, in this case *papers*, we prepare them for merging.

When the refactoring step is complete, we have five generated candidate service classes, each with the same name, and each with its own set of candidate service operations. We then merge the candidate service operation functions from the five different classes into one single class file containing all of the candidate service operations, as shown in Figure 9. If the different generated candidate service classes have two operations with the same name and functionality, then we merge them by hand into a single operation function. If their functionality is different, then we must rename one of them and its corresponding calls in the adapted page file.

As part of the refactoring transformation, the results required by each candidate service operation are analyzed and a result value class generated for each candidate service operation. These classes do not require merging since each is a unique separate class, but the files containing them are merged into one, simply by concatenating them.

```
<?php
include_once("functions.php");
<markIncludes/>
include_once("header.php");
////////// (... 10 lines elided ...) //////////
if (isset($_GET['paper_id'])) {
   $id = (int) $_GET['paper_id'];
   <service function=getPaperDetails>
   $result = query("SELECT title, abstract, session_id, pdf, pdfname FROM papers WHERE paper_id='".$id."'");
   $title = $result[0]["title"];
   $abstract = $result[0]["abstract"];
   $session_id = $result[0]["session_id"];
   $pdf = $result[0]["pdf"];
   $pdfname = $result[0]["pdfname"];
   $result = query("SELECT user_id FROM authors WHERE paper_id='".$id."' ORDER BY `order`");
   $authors = Array();
   if ($result){
      foreach($result as $row) {
         $authors[] = $row;
      }
   }
   </service>
}
////////// (... 60 lines elided ...) //////////
   include("editform3.php");
   <service function=getFileEdit>
   $result = query("SELECT name, data FROM files WHERE paper_id='".$id."'");
   </service>
////////// (... 75 lines elided ...) //////////
   if (!isset($_POST['paper_id'])) {
         // new paper
      <service function=addPaper>
      $row = query ("SELECT MAX( `order` ) as max FROM `papers` WHERE session_id = '".$session."'");
      $order = (int) $row[0] + 1;
      query ("INSERT INTO papers (title, abstract, pdf, pdfname, session_id, `order`) VALUES ('".$title."', '".$
            abstract."', '".$pdf."', '".$pdfname."', '".$session."', '".$order."')");
      $row = query ("SELECT paper_id FROM papers WHERE title='".$title."' AND abstract='".$abstract."' AND pdfname
            ='".$pdfname."' AND session_id='".$session."' ORDER BY paper_id DESC");
      $id = $row[0]['paper_id'];;
      </service>
   } else {
         // updated paper paper
      if (!empty($filename)) {
         $pdfSetString = "pdf='$pdf', pdfname='$pdfname',";
      } else {
         $pdfSetString = "";
      }
      <service function=updatePaper>
      query("UPDATE papers SET title='".$title."', abstract='".$abstract."', ".$pdfSetString." session_id='".$
            session."' WHERE paper_id='".$id."'");
      $id = (int) $_POST['paper_id'];
      query("DELETE FROM authors WHERE paper_id='".$id."'");
      </service>
   }
////////// (... 50 lines elided ...) //////////
   $num = 0;
   <service function=addAuthors>
   foreach ($_POST['authors'] as $author) {
      if (! empty($author)) {
         query ("INSERT INTO authors (`paper_id`, `user_id`, `order`) VALUES ('".$id."', '" .
               mysql_real_escape_string($author) . "', '".$num."')");
      }
      $num++;
   }
   </service>
   if (!isset($_POST['paper_id'])) {
      print "Paper added successfully";
   } else {
      print "Paper updated successfully";
   }
   print ". View <a href='showpaper.php?paper_id=$id'>the paper</a>";
}
include_once("footer.php");
?>
```

Fig. 8. Paper Management Operation Markup in the SCARF *editpaper.php* page

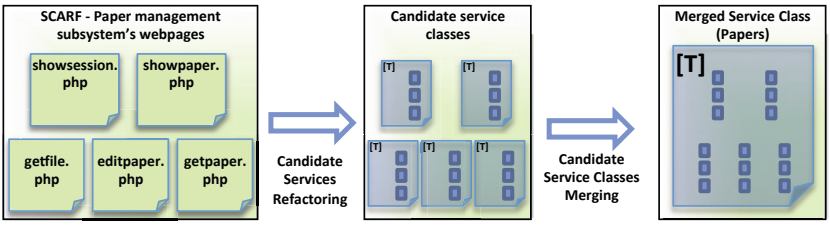

Fig. 9. Generating and Merging Candidate Service Operations from Multiple Application Pages

4.4 Step 3: Type Inference

Once the generated candidate service classes for each page have been merged into a single merged candidate service class, the remaining steps of the process simply proceed as for a single page. We use the the dynamic type inference technique of Section 3.3 to infer the types of the operation parameters of the new merged candidate service class by instrumenting and running the class with the adapted application pages to gather and store dynamic type information, and then use the type merging transformation to add the inferred types to the merged candidate service class operation functions.

Figure 10 shows the merged SCARF paper management candidate service after the instrumentation transformation, with instrumentation code highlighted. This temporary instrumented version of the merged candidate service class is exercised by running the SCARF application with the adapted application pages, exploring all of the paper management related links from the SCARF user interface until all of the candidate service operation functions have been called at least once.

The output of this step is an instrumentation file containing type signatures for all of the parameters of all fourteen of the candidate service operation functions (Figure 11), which are then merged into the candidate service class using the typing transformation described in Section 3.3 to yield the fully typed merged candidate service class.

4.5 Step 4: Conversion to SCA

In the final stage of the automated migration, conversion to an SCA service component, we use the transformations of Section 3.4 to turn the SCARF candidate service class into an SCA-based web service, and modify the pages of the adapted SCARF web application to use the new service as a client.

1. The typed candidate service class is automatically transformed to remove NULL parameters, to insert code to unserialize parameters that are of type object or array, and to serialize the result object of each operation.
2. The SCA annotation transformation of Section 3.4 is applied to the serialized candidate service class to yield an SCA service component. We add an include statement for the SCA library, and SCA annotations for the class and methods. These include *@service* and *@binding.soap* annotations for the class, and parameter and result type annotations for each operation function. This enables the class as a service.

```
<?php
include_once ("functions.php");
include_once "papers_return.php";

class papers {
   function addAuthors ($_POST, $author, $id, $num) {
      $FileHandle = fopen ("/tmp/papers.merge.php", 'a');
      fwrite ($FileHandle, "class papers{\n function addAuthors(");
      fwrite ($FileHandle, gettype ($_POST).' $_POST');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($author).' $author');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($id).' $id');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($num).' $num');
      fwrite ($FileHandle, ") {\n");
      fwrite ($FileHandle, " }\n}\n");
      fclose ($FileHandle);
      foreach ($_POST ['authors'] as $author) {
         if (! empty ($author)) {
            query ("INSERT INTO authors (`paper_id`, `user_id`, `order`) VALUES ('".$id."', '".
                  mysql_real_escape_string ($author)."', '".$num."')");
         }
         $num ++;
      }
      return new addAuthors_return ();
   }

   function updateFile ($id, $oldname, $name, $ext, $type, $data) {
      $FileHandle = fopen ("/tmp/papers.merge.php", 'a');
      fwrite ($FileHandle, "class papers{\n function updateFile(");
      fwrite ($FileHandle, gettype ($id).' $id');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($oldname).' $oldname');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($name).' $name');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($ext).' $ext');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($type).' $type');
      fwrite ($FileHandle, ",");
      fwrite ($FileHandle, gettype ($data).' $data');
      fwrite ($FileHandle, ") {\n");
      fwrite ($FileHandle, " }\n}\n");
      fclose ($FileHandle);
      query ("DELETE FROM files WHERE paper_id='".$id."' AND name='".$oldname."'");
      query ("INSERT INTO files (paper_id, name, ext, type, data) VALUES ('".$id."$', '".$name."', '".$ext."',
         '".$type."', '".$data."')");
      return new updateFile_return ();
   }

   ////////// (12 more instrumented candidate service operation functions) //////////
}
?>
```

Fig. 10. Instrumented Merged Candidate Service Class for SCARF Paper Management

3. Invoking the converted service class using the SCA WSDL generation URL $http://hostname/path/papers.php?wsdl$ causes the SCA platform to generate the WSDL service description for the new service from the SCA annotations.
4. In the final transformation, the adapted source pages of the SCARF web application are converted to be an SCA client of the new web service. We add the include statement for the SCA library, create an instance of the proxy object for the service, and update each service operation call to use it.

Figure 12 shows the final SCARF paper management service class after conversion to an SCA service. Each of the adapted SCARF application pages from which the service operations were extracted are converted to SCA WSDL clients of the service using the final transformation of Section 3.4, and the migration is complete.

```
<?php
class papers{
  function getPaperDetails(array $result,integer $id,string $title,string $abstract,NULL $session_id,string $pdf,
        string $pdfname,NULL $authors,array $row) { }
  function getFileEdit(array $result,integer $id) { }
  function updatePaper2(string $newname,integer $id,string $oldname) { }
  function addPaper(array $row,integer $session,NULL $order,string $title,string $abstract,string $pdf,string $
        pdfname,integer $id) { }
  function updateFile(string $id,string $oldname,string $name,string $ext,string $type,string $data) { }
  function updatePaper(string $title,string $abstract,string $pdfSetString,integer $session,integer $id,array $_POST
        ) { }
  function deletePaper(integer $id,string $oldname) { }
  function addAuthors(array $_POST,NULL $author,integer $id,integer $num) { }
  function getFile(NULL $id,array $_GET,NULL $name,NULL $result) { }
  function getpaper(NULL $id,array $_GET,NULL $result) { }
  function getPaperAttribs(integer $id,array $_GET,array $result,NULL $title,NULL $abstract) { }
  function getFileInfo(NULL $result2,string $id) { }
  function paperTitle(NULL $result2,array $row) { }
  function paperAuthor(array $result3,array $row2) { }
}
?>
```

Fig. 11. Type Instrumentation Output of the SCARF Candidate Service Class

4.6 SCARF / SOA: Testing the Result

We validated the conversion of the SCARF paper management subsystem into a web service by testing the migrated SCARF web application in two ways.

First, we already knew how to cover all of the new web service operations from the SCARF browser interface, because we already had to test all of the operations of the candidate service class from the web interface as part of the type inference instrumentation step. To test the migrated SCARF, we exercised all of the same links in the SCARF user interface to cover all of the operations of the new paper management web service, and verified that the behaviour and output of each of the pages was the same for these tests in both the original and the migrated web application.

Second, to be certain that we had not changed any hidden behaviour, we logged the values of PHP variables before and after each tagged candidate service operation code segment in the original application, and compared those values to the same variables before and after the calls to the corresponding web service operations of the new extracted SCARF paper management web service.

5 Related Work

Our approach does not attempt to solve the identification of services, rather we leverage the results of other research such as the work done by Asuncion et al. [4], which uses a goal-based, model-driven approach to identify business rules in the application.

There has been a lot of work on migration of traditional legacy systems to SOA. Lewis et al.'s [8] SMART process provides a set of guidelines to identify the context, current system and target SOA system states and the gaps between them, and suggests the steps required to create a migration strategy. O'Brien et al. [14] describe a strategy for architecture reconstruction in legacy systems by identifying and reusing legacy components as services. Zhang and Yang introduce the use of cluster analysis [15], and Dwivedi and Kulkarni present a model-driven approach for service identification which utilizes process maps and service hierarchies [16]. Other approaches are presented by Chen et al. [17] and Aversano et al. [18].

```php
<?php
include_once ("functions.php");
include_once "papers_return.php";
include "SCA/SCA.php";

/**
   * @service
   * @binding.soap
   */
class papers {
   /**
     * @param string $_POSTStr
     * @param integer $id
     * @param integer $num
     * @return string
     */
   function addAuthors (string $_POSTStr, integer $id, integer $num) {
      $_POST = unserialize ($_POSTStr);
      foreach ($_POST ['authors'] as $author) {
         if (! empty ($author)) {
            query ("INSERT INTO authors (`paper_id`, `user_id`, `order`) VALUES ('".$id."', '".
                  mysql_real_escape_string ($author)."', '".$num."')");
         }
         $num ++;
      }
      return serialize (new addAuthors_return ());
   }
   /**
     * @param string $id
     * @param string $oldname
     * @param string $name
     * @param string $ext
     * @param string $type
     * @param string $data
     * @return string
     */
   function updateFile (string $id, string $oldname, string $name, string $ext, string $type, string $data) {
      query ("DELETE FROM files WHERE paper_id='".$id."' AND name='".$oldname."'");
      query ("INSERT INTO files (paper_id, name, ext, type, data) VALUES ('".$id."$', '".$name."', '".$ext."', '".$
            type."', '".$data."')");
      return serialize (new updateFile_return ());
   }
   /**
     * @param string $newname
     * @param integer $id
     * @param string $oldname
     * @return string
     */
   function updatePaper2 (string $newname, integer $id, string $oldname) {
      query ("UPDATE files SET name='".$newname."' WHERE paper_id='".$id."' AND name='".$oldname."'");
      return serialize (new updatePaper2_return ());
   }
   /**
     * @param integer $id
     * @param string $oldname
     * @return string
     */
   function deletePaper (integer $id, string $oldname) {
      query ("DELETE FROM files WHERE paper_id='".$id."' AND name='".$oldname."'");
      return serialize (new deletePaper_return ());
   }

   ////////// (10 more service operations) //////////
}
?>
```

Fig. 12. Final Migrated SCARF Paper Management Service Class

Much less work has been done in the area of migrating web applications to SOA. Tatsubori and Takashi's H2W framework [19] constructs web service wrappers for existing multi-paged web applications, and Dezhgosha and Angara [20] discuss how web services can be used to leverage existing web applications in a similar way. Vijaya and Rajan [21] focus on exploring the benefits of converting to web services, and Ajlan and Zedan [22] have worked on exposing the assignment module of Moodle as a web service, using a UML collaboration diagram to analyze and capture the necessary features.

In contrast, our work proposes a concrete generic framework of iterative steps for the migration of identified functionality to web services. Our goal is automation, and we

have implemented our framework as a source transformation-based toolset that largely automates the migration of identified service operations in legacy PHP web applications to SCA-based web services.

6 Conclusions and Future Work

In this paper we have presented a framework and tool prototype that automates the migration of monolithic PHP web applications to web services in an SOA environment. The framework 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 monolithic web applications to SOA, and the first to describe a complete detailed process with significant levels of automation.

Our framework consists of several automated steps: candidate service refactoring, candidate service separation, parameter type inference, service component conversion, and database refactoring. The result of applying our process is a new web application in which identified business operations have been separated into web services that both serve the original web application and can be reused by other applications.

At present our prototype implementation does not handle every feature of the PHP language. In particular, the refactoring step does not always detect all modifications or uses of variables in the tagged candidate service code fragments, in particular when variables appear inside strings. As a result the inferred parameters and return classes may in some unusual cases be incomplete. However, this is a well understood problem and it is relatively straightforward to extend the implementation. Due to the use of the TXL source transformation engine and its PHP grammar, at present our source transformations do not retain PHP comments from the original code. This is a known difficulty with source transformation tools, and can be addressed using the techniques described in Malton et al. [23].

There are several future lines of research for our work. While our migration process presently uses serialization to transfer non primitive data types, further analysis of the client application and the candidate service class could provide automated assistance for the migration of core data structures to Service Data Objects (SDO-DAS-XML) [24]. Currently every identified code segment in the original application is converted into a separate service operation. Clone detection techniques could identify similar operations and merge them into a single operation. While we have illustrated the automation of our process on the PHP language, our framework and its steps are not specific to any particular language. Extending our prototype automated migration tools to other web application languages such as Python is another area for future research.

References

[1] Tatsubori, M., Takahashi, K.: Decomposition and abstraction of web applications for web service extraction and composition. In: ICWS, pp. 859–868 (2006)
[2] Rajan, A., Otieno, J.: Leveraging traditional distributed applications to web services for e-learning applications. In: DEXA, pp. 430–435 (2004)
[3] Dezhgosha, K., Angara, S.: Web services for designing small-scale web applications. In: EIT, 4 p. (2005)

[4] Asuncion, C.H., Iacob, M.E., van Sinderen, M.: Towards a flexible service integration through separation of business rules. In: EDOC, pp. 184–193 (2010)
[5] Achour, M., Betz, F., Dovgal, A., Loopes, N., Magnusson, H., Richter, G., Seguy, D., Vrana, J.: PHP Manual, http://www.php.net/manual/en/index.php (last accessed August 2011)
[6] Van Rossum, G.: Python programming language, http://www.python.org/ (last accessed August 2011)
[7] Smith, D.: Migration of legacy assets to service-oriented architecture environments. In: ICSE, pp. 174–175 (2007)
[8] Lewis, G., Morris, E., O'Brien, L., Smith, D., Wrage, L.: SMART: The service-oriented migration and reuse technique. In: STEP, pp. 222–229 (2005)
[9] Sneed, H.M., Sneed, S.H.: Creating web services from legacy host programs. In: WSE, pp. 59–65 (2003)
[10] Almonaies, A., Cordy, J.R., Dean, T.R.: Legacy System Evolution towards Service- Oriented Architecture. In: SOAME, pp. 53–62 (2010)
[11] Cordy, J.R.: The TXL source transformation language. Sci. Comput. Program. 61, 190–210 (2006)
[12] Moodle Trust: Moodle, http://Moodle.org (last accessed October 2010)
[13] Tarjan, P., McKeown, N.: The Stanford Conference and Research Forum, http://scarf.sourceforge.net/ (last accessed March 2013)
[14] O'Brien, L., Smith, D.B., Lewis, G.A.: Supporting migration to services using software architecture reconstruction. In: STEP, pp. 81–91 (2005)
[15] Zhang, Z., Yang, H.: Incubating services in legacy systems for architectural migration. In: APSEC, pp. 196–203 (2004)
[16] Dwivedi, V., Kulkarni, N.: A model driven service identification approach for process centric systems. In: Congress on Services Part II, SERVICES-2, pp. 65–72 (2008)
[17] Chen, F., Li, S., Chu, W.C.C.: Feature analysis for service-oriented reengineering. In: APSEC, pp. 201–208. IEEE Computer Society (2005)
[18] Aversano, L., Cerulo, L., Palumbo, C.: Mining candidate web services from legacy code. In: WSE, pp. 37–40 (2008)
[19] Tatsubori, M., Takashi, K.: Decomposition and abstraction of web applications for web service extraction and composition. In: ICWS, pp. 859–868 (2006)
[20] Dezhgosha, K., Angara, S.: Web services for designing small-scale Web applications. In: International Conference on Electro Information Technology, 4 p. (2005)
[21] Rajan, A.V.S., Otieno, J.: Leveraging traditional distributed applications to web services for e-learning applications. In: 15th Intl. Workshop on Database and Expert Systems Applications, pp. 430–435 (2004)
[22] Ajlan, A., Zedan, H.: E-learning (MOODLE) Based on Service Oriented Architecture. In: The EADTU's 20th Anniversary Conference, pp. 62–70 (2007)
[23] Malton, A.J., Schneider, K.A., Cordy, J.R., Dean, T.R., Dousineau, D., Reynolds, J.: Processing software source text in automated design recovery and transformation. In: IWPC, pp. 127–134 (2001)
[24] Charters, G., Peters, M., Maynard, C., Srinivas, A.: An introduction to Service Data Objects for PHP, http://www.ibm.com/developerworks/library/os-sdophp/ (last accessed July 2011)