A Software Support to Initiate Systems Engineering Students in Service-Oriented Computing

CRISTIAN MATEOS, MARCO CRASSO, ALEJANDRO ZUNINO, MARCELO CAMPO

ISISTAN *Research Institute*, UNICEN *University*, *Campus Universitario*, *Tandil* B7001BBO, *Buenos Aires*, *Argentina*

Received 31 August 2010; accepted 1 April 2011

ABSTRACT: An evolutionary process that is currently taking place in the software industry is the shift from developing applications from scratch to discovering and assembling services published in the Internet. This has given birth to a new computing paradigm called service-oriented computing (SOC). We investigated whether simplifying and automating tasks inherent to SOC-based development, while exploiting systems engineering students' experience in earlier paradigms, namely object orientation, reduce the cognitive effort needed to learn SOC. The study involved 38 undergraduate students plus 7 postgraduate students from 4 universities, which attended a course about SOC development models and technologies. Then, they were asked to develop a real-life service-oriented application using two alternatives, namely existing SOC libraries and a software support of our own named EasySOC. EasySOC promotes using common object-oriented design patterns to structure service-oriented applications, facilitates service discovery, and hides many technological details from users. The students were surveyed about their perception on both alternatives using a Likert-based questionnaire. Results show that the students, who had no previous experience in service-oriented notions before the experiment, perceived that EasySOC allows focusing on essential aspects of the paradigm while concealing accidental aspects, and provides adequate support and guidance to bridge the gap. © 2011 Wiley Periodicals, Inc. Comput Appl Eng Educ View this article online at wileyonlinelibrary.com/journal/cae; DOI 10.1002/cae.20551

Keywords: service-oriented computing; teaching/learning strategies; programming and programming languages; authoring tools and methods

MOTIVATION AND PROBLEM STATEMENT

A computing paradigm refers to the set of concepts, principles, and methods for expressing computation that allows a human to command, through a software application, a computer to perform a set of given tasks. Service-oriented computing (SOC) is a contemporary computing paradigm that supports the development of applications that are built by composing existing distributed pieces of functionalities termed *services* [1]. Services, in turn, are published and accessed through network-aware protocols. From a software engineering standpoint, SOC is an interesting paradigm for application development, since it heavily promotes service reuse to rapidly construct end user applications [2]. In other words, the core idea is not to manually implement applications from scratch but partially rely on and invoke already implemented, network-accessible external pieces of software. From a technological perspective, SOC is not only interesting, but also challenging since it requires the handshaking between user applications and services to be distributed and interoperable. In SOC terminology, ‘‘interoperable’’ refers to the ability of a third-party service to be effortlessly used from different applications and software platforms.

Certainly, SOC is not simply another way of designing and developing applications, but it is conceived as a revolutionary paradigm together with those paradigms that have historically predominated in software development and therefore computer science education up to now [3]. Such paradigms include imperative programming, which was developed in the 1950s, procedural programming, which became particularly popular during the 1970/1980s, and object-oriented programming, whose inception took place in the 1980s.

The high complexity of today's software systems has made the SOC paradigm one of the most valuable tools for software engineers and practitioners. Many software vendors have already embraced SOC for building applications and its popularity is relentlessly gaining momentum. In the academia, there is a great consensus about the fundamental role that SOC concepts must play in the CV of computer science students [4]. These two facts have motivated the inception of SOC courses

Correspondence to C. Mateos (cmateos@conicet.gov.ar).

in universities worldwide, and even high schools [3,5], which constitutes an effort to meet the ever-increasing demand for higher and continuous education in software engineering. Unfortunately, though most of the time adopting a new computing paradigm comes at the expense of a very costly ‘‘paradigm shift,’’ little attention has been paid to such a new reality in the academia from a pure educational perspective. Plainly, paradigm shift means the act of radically changing the way the constituent elements of software systems are combined and organized [6].

An exhaustive literature review yielded as a result that, to date, just a few approaches aimed at teaching SOC in Systems Engineering programs have been proposed, being Water [7] and WS^{EXP} [8] the most representatives. Moreover, the weak point of these two specific approaches is that they only capture an incomplete fraction of the fundamental elements of the paradigm. Roughly, they mostly focus on teaching service-oriented technologies by paying little or even no attention to essential aspects of SOC design that relate to the activities of consuming services from within applications as well as exposing services to other applications. Consuming a service is the task of including explicit calls to a service within a user application, which in a broad sense is similar to importing existing code libraries and performing invocation to their functions. Exposing a service, on the other hand, refers to the task of publishing or making a service accessible through a network so that other applications can consume it. Furthermore, there are related efforts, such as the work by Wu et al. [9,10], who use SOC technologies for building educational software; however this software is not designed to teach SOC but to assist students in learning mechanical and electronic engineering concepts. A similar approach is taken by GridFoRCE [11], a software platform for teaching Grid Computing [12] that is implemented via service-oriented technologies.

To sum up, as far as we know there is a lack of approaches to effectively teach the above-mentioned aspects. Then, we are facing as teachers the need of newer and more integral tools to convey the fundamentals of this contemporary paradigm. However, one of the most challenging issues associated with learning and teaching SOC is the plethora of software technologies surrounding and materializing the paradigm, which often eclipse the simplicity of the concepts underpinning it. This also applies, to some extent, when teaching traditional Web programming and development. Therefore, by just relying on a subset of such technologies, one cannot guarantee that all the essential aspects of SOC design are made explicit and exercised.

The rest of the article is organized as follows: The next section gives an overview of our approach to teaching SOC and its associated software materialization called EasySOC. In addition, the section briefly describes the research hypothesis that arises as a consequence of our approach and the methodology used to provide experimental evidence about its validity. Then, the Designing and Developing Service-Oriented Applications With EasySOC Section presents the EasySOC software in detail. The section explains the principles underpinning EasySOC, discusses some implementations issues, and illustrates its usage with a case study. Later, the Evaluation of EasySOC Section reports the experimental evaluation of our approach from an experience with the aforementioned students and the EasySOC tool in the context of a real SOC course. Finally, the lost section concludes the article and points out lessons learned.

APPROACH AND RESEARCH HYPOTHESIS

Nowadays, an SOC application is thought as a collection of Web Services [13], distributed programs with well-defined interfaces that can be located and invoked via popular Web protocols such as HTTP, FTP, or more recently SOAP [14]. Upon reusing a Web Service in a user application, a developer first retrieves the services he needs from a public registry, and then uses the associated protocol-specific libraries for calling the operations or functions of these services. One illustrative example is the Google’s Search Web Service [15], a service-based interface to the same search functionality an end user can access by using a regular Web browser. The service offers for instance operations for googling the Web or spell checking text from within any kind of application apart from the browser.

The architectural model underlying Web Services encompasses three elements: service providers, service requesters, and service registries. Basically, a service provider creates a Web Service description by employing WSDL (http://www.w3.org/TR/wsdl), a language for describing the interface—i.e., the offered operations—of Web Services, and publishes it in a service registry using UDDI (http://uddi.xml.org), a standard service repository for publishing and discovering services. Service requesters, or application developers, use the registry to find Web Services that match their functional needs, and then use the corresponding WSDL descriptions to invoke operations. As a consequence, developers do not re-implement existing services but reuse these latter instead.

Even when this model may appear intuitive at first sight, mastering it is indeed more challenging compared to learning well-established programming paradigms such as object orientation, which in turn also reuse concepts from even older paradigms. Particularly, any object-oriented application consists of a number of objects that communicate between each other via regular method (i.e., functions) calls. By drawing a parallel with SOC, a service-oriented application also comprises a number of components that interact between each other via message exchange. However, SOC applications present a number of distinctive characteristics regarding component/application construction and message handling, namely:

- Unlike classes, in which having interfaces explicitly declared for them is totally optional, a single service always has at least two artifacts associated: an interface specification in WSDL and its implementation, which conforms to this specification, in a conventional programming language. In this sense, building SOC applications consuming services requires to understand yet another interface specification language and data type system.
- By nature, an SOC application is *distributed*, since some components may perform calls to services that physically reside on different machines. Most object-oriented applications, on the other hand, comprise objects that are installed in the same machine, which makes common development tasks such as application testing and debugging location-unaware and hence simpler.
- Related to the previous issue, services must be contacted by using remote messaging protocols. Moreover, the spectrum of protocols and technologies implementing them is rather wide, and so are the specifics of each choice, which must be apprehended. With object orientation, on the

other hand, there is no need of remote protocols since application objects communicate via traditional, local method calls.

- Class (and object) assembling upon building an application is mostly done at development time by selecting the specific set of classes that will be used to implement the desired behavior. Moreover, some of these classes are usually implemented from scratch, while others are reused by importing external class libraries. In SOC applications, Web Services play the role of ''class libraries'' that can be used as building blocks for new applications. However, public Web Services live in an inherently massively distributed environment, and as such there are many services providing similar behavior. In this sense, users must browse huge service registries before finding the specific services they need for their applications, which apart from requiring more efforts, can be counterintuitive for an adopting user.

In this light, we claim that for SOC teaching to become more effective, there is a need for a new tool that allows students to capture the three elements of the Web Services model while still learning the main technologies materializing this model at the correct level of abstraction. Indeed, using intuitive and rich GUIs has proven to be a viable and effective approach to teaching in engineering educational environments [16]. Moreover, the idea has been particularly successfully applied in teaching object-oriented programming [17–19]. Therefore, our goal is not come out with yet another graphical tool for teaching object orientation, but reusing this approach for teaching the SOC paradigm.

Similarly to the aforementioned past studies about tools for learning the object-oriented paradigm, the new tool should hide to some extent the SOC paradigm challenges listed above as much as possible from users. We propose EasySOC, a Java-based software tool to simplify the construction of SOC applications by hiding many technological details behind an intuitive development environment. Unlike related efforts, EasySOC takes an application-centric approach to SOC that allows students to gradually explore the process of reusing external services. In addition, EasySOC supports the easy creation of Web Services and the administration of registry-related information. Moreover, EasySOC has been implemented as a plug-in of Eclipse (http://www.eclipse.org), a very popular development environment. Eclipse was originally created by IBM in November 2001, but it became open-source in 2004. From that moment on, Eclipse has gained much popularity among users because it constitutes a free software platform comprising extensible tools for building, deploying, and managing applications. This feature makes EasySOC not only an educational tool for SOC courses, but also a potential development platform supporting the SOC paradigm, which eventually may be adopted by software engineers to manage the life cycle of SOC applications. EasySOC can be downloaded from http://sites.google.com/site/easysoc.

We have assessed the benefits of EasySOC through a controlled learning scenario in the context of an SOC course with 45 participants including 38 last-year systems engineering students, and 7 postgraduate students (PhD candidates in Computer Science with Systems Engineering background) from 4 different universities of Argentina. These students were involved in the elaboration of a two-phase homework, which consisted on developing the same SOC application by using both traditional Web Service development software libraries of their choice and EasySOC. Then, we asked all the students to complete an online survey (http://grid.isistan.exa.unicen.edu.ar:8080/encuestaSOC, in Spanish) so as to collect their opinions about the whole experience.

We worked on the hypothesis that EasySOC sharpens the learning curve needed to build well-structured service-oriented applications provided students have some basic concepts from object-oriented programming (i.e., inheritance, composition, etc.), and the SOC paradigm itself, which were given in a lecture-based style. This hypothesis arises as a consequence of the principles behind the design goal of EasySOC, which is to raise the level of abstraction at which the essential elements of SOC applications are modeled and designed but without losing flexibility to select the associated enabling technologies. The obtained results from analyzing the students' opinions suggest that most of the respondents perceived that EasySOC is indeed a convenient and an intuitive tool for designing and implementing service-oriented applications. Since the students had very good programming skills but not much knowledge on SOC development before the experiment, which is in fact the initial state of most last-year students of BSc programs and first-year students of PhD programs, we can reasonably extrapolate these results to argue that EasySOC may be useful to teach SOC-based development in similar classroom situations.

The next section presents the EasySOC software support from a conceptual as well as a technical perspective.

DESIGNING AND DEVELOPING SERVICE-ORIENTED APPLICATIONS WITH EasySOC

EasySOC is a tool that prescribes an easy methodology to design service-oriented applications and guide users during the entire life cycle of their software. Metaphorically, central to this methodology is to think of service-oriented applications as special puzzles. Such special puzzles have two types of pieces. One type of pieces represents the *internal* components of a service-oriented application (the ones implemented by users), whereas another stands for third-party services (the ones not implemented but discovered and reused). Hence, service pieces have some peculiarities. First, they are public and as such they can be used to solve many puzzles, that is, called from different applications. Second, there are many service pieces with the same content, so the puzzle solver—in this case a developer—should select among the available alternatives. Third, the shape of service pieces can be slightly modified, or adapted, to fit into a puzzle without affecting those puzzles already using them. Here, the shape represents the interface with the operations offered by a service piece. Figure 1 illustrates this metaphor. Internal component pieces and service pieces are depicted using gray and black, respectively.

The approach taken by EasySOC to build these special puzzles is to start by joining gray pieces. Once the gray parts of the puzzle are built, the solver should look for every hole in the ''picture,'' and pick a proper public piece to fill it. This should be iteratively applied to associate a black piece with each hole in order to complete the picture. Then, when building a service-oriented application with EasySOC, a developer thinks of such an application as a collection of internal components invoking external ones, that is, services. Having in mind SOC

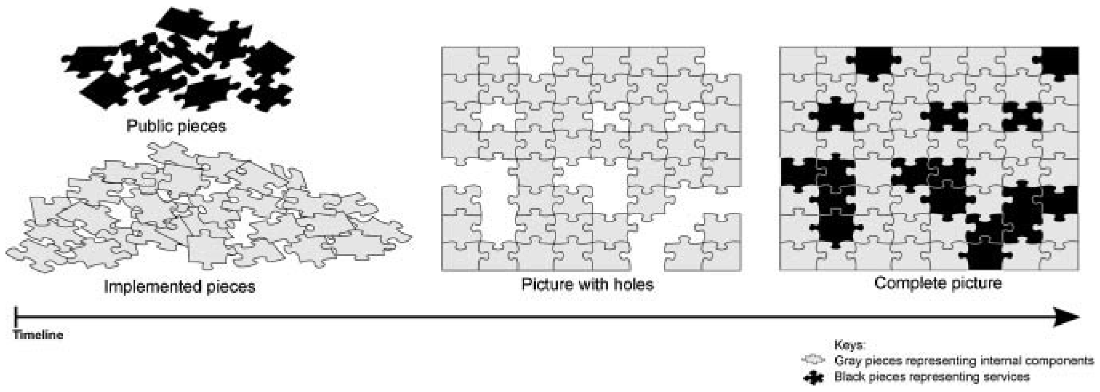

Public pieces
Implemented pieces
Picture with holes
Complete picture
Timeline
Keys:
Gray pieces representing internal components
Black pieces representing services

Figure 1 Conceptual service-oriented puzzle.

applications as arrangements of internal components and services, EasySOC encourages developers to first design, implement, and test for correctness the internal components of their applications, and then discover and incorporate services into them. This is analogous to first arrange gray pieces together, and in turn fill the holes of the resulting picture using black ones.

There are, naturally, many similar ways of designing and implementing the pieces or components of an application. However, not necessarily all the alternatives to arrange internal components and services are viable. For instance, several researchers have shown that the alternative adopted by common libraries for invoking Web Services misleads developers to build service-oriented applications that are rather hard to understand and to maintain [20]. Unlike these libraries, EasySOC proposes a programming methodology to arrange the components and services of an SOC application that facilitates their maintenance afterward [21]. This is achieved by raising the level of abstraction by which the necessary plumbing is done.

At design time, users employing EasySOC represent an individual functionality planned to be delegated to a service as an abstract interface, which is analogous, for example, to a C library header and as such specifies the signatures of the operations needed by a user. In consequence, users produce incomplete applications, in which some of their constituent components are implemented, and those intended to be outsourced to services are abstractly represented. In order to complete an application, a user should associate an existing, concrete service to each defined abstract interface. In this sense, the user should look for available services in a registry, and select one candidate.

With EasySOC, associating—also called *binding*—a third-party service to an abstract interface requires to add two components into an application. First, it is necessary to use the Proxy object-oriented construct to build a component that provides to internal components an identical interface to that of the called service. Such proxy is responsible for forwarding through the network all operation requests coming from internal components to the corresponding running Web Service [22]. Then, internal components can invoke a service via a proxy component regardless of the network location of the associated service.

One implication of having the functionality of services represented as abstract interfaces previous to discover them is that service interfaces and therefore proxy interfaces may differ from abstract ones. For instance, let us suppose that a proxy operation getCalendarHolidays, which receives as input an integer representing a month and returns a list of integers, has been discovered and selected to fill the ‘‘hole left’’ by an abstract interface designed to return an array of floats. Under these situations in which actual and abstract service interfaces differ, EasySOC proposes to use the Adapter notion from object-oriented programming to build an extra software component that bridges the differences. An adapter is responsible for performing type conversions and resolving any operation signature mismatch found between actual and abstract service interfaces [22]. This has been shown as a good design practice, since by decoupling internal components from specific service interfaces, applications can be easily accommodated to support service replacements. This is the situation, for example, when an application using the Google’s Search Web Service is modified to use a similar service but offered by a different provider (e.g., Microsoft’s Bing Web Service, http://msdn.microsoft.com/en-us/library/cc980922.aspx).

The final step prior to the incorporation of a selected service into a target application is to fit together the internal components, proxies, and adapters. To do this, EasySOC uses another object-oriented design pattern called dependency injection (DI) [23]. DI is a technique for supplying an external dependency to a software component, in which the process of obtaining the needed dependency is performed by a special entity called the DI container. As shown in Ref. 21, an interesting implication of using DI in SOC is that the source code of the internal components of a service-oriented application can be isolated from the details for obtaining and invoking services (e.g., the Web pointer to WSDL documents, invocation protocols, etc.). Then, a user thinks of a third-party service as any other regular component providing a clear interface to its operations. To clarify this idea, Figure 2 depicts the anatomy of an EasySOC-based SOC application, using the UML 2.0 notation for modeling software components. In short, for developing SOC applications, EasySOC promotes a decoupled yet ordered mechanism for assembling components and services together.

Figure 2 shows that when a user wants to call an external service S with interface I_S from within for instance internal components C' and C'', at design time a dependency among these two latter and S is indirectly established via an abstract interface, which may differ from I_S. In this context, S may be the abovementioned Web Service for returning calendar holidays, and I_S its actual interface with a int[]

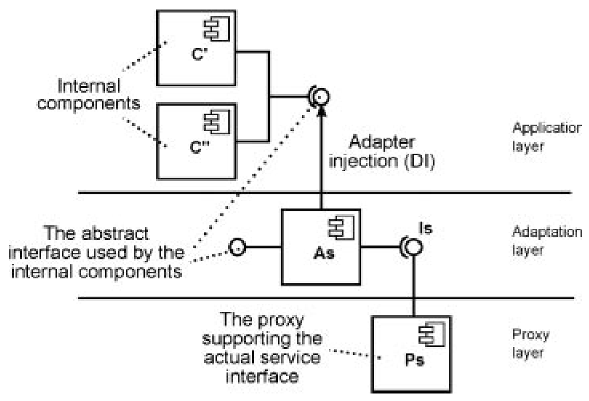

C'
C''
Internal components
Adapter injection (DI)
Application layer
The abstract interface used by the internal components
As
Is
Adaptation layer
The proxy supporting the actual service interface
Ps
Proxy layer

Figure 2 Anatomy of an SOC application produced with EasySOC.

getCalendarHolidays(int) operation. This kind of dependency is commonly managed by a DI container that ‘‘injects’’ into C' and C'' an adapter (let us say A_S) that wraps a proxy to S (let us say P_S). Then, at run-time the code of the internal components will end up calling any of the methods declared in P_S through A_S, which transparently invokes the remote service through P_S. We refer as A_S to the service adapter that accommodates the actual interface of S to the interface expected by internal components specified early by the programmer, in our example including a float[] getCalendarHolidays(int) operation.

Interestingly, this mechanism is almost transparent and does not demand coding effort from the user, as it only requires associating a configuration file to the SOC application, which is read by the DI container to determine to which internal components adapters should be injected. With the EasySOC development model, service-oriented applications are free from code for configuring and using Web Service protocols, since the code for contacting the service is isolated beneath the application layer (see Fig. 2), and the corresponding configuration parameters are placed on a separate file, which is processed at run-time by the DI container. This produces a better code in terms of the level of isolation from SOC-related technologic details, thus users can rapidly focus on SOC-specific modeling issues.

Despite the positive aspects of the programming methodology proposed by EasySOC in terms of decoupling internal components and Web Services, EasySOC relies on replacing manual coding by introducing a number of new development tasks, particularly discovering services, adapting service interfaces, and assembling adapters into internal components. This might involve, unless properly supported in the tool, a learning curve for novice SOC users such as students. Even when we have proved such learning to be steeper than that of traditional models for SOC programming [20], it nevertheless involves some knowledge and effort.

To overcome these costs, we have built an Eclipse plug-in that aims at automatically performing these tasks on behalf of SOC application programmers. The tool exploits the concept of Query-By-Example for Web Services described in Ref. 24. This concept suggests that due to the structure inherent to service-oriented applications and Web Service descriptions in WSDL, the ‘‘shape of a service piece’’ or abstract interface can be seen as an example of what a user is looking for. This is built on the fact that with the WSDL language, service publishers can describe their services as object-oriented interfaces, with methods and arguments as well. Basing on the Query-By-Example concept, the tool gathers certain information that is implicitly conveyed in the source code of expected or abstract service interfaces. Gathered information is preprocessed to build a refined textual description of users’ needs. Accordingly, an effective query is generated provided that programmers followed documenting and naming best practices in their service-oriented applications. This is because the query generation heuristic gathers relevant terms from the names and comments of an interface, its methods and arguments. Finally, the query is sent to a registry that matches the information gathered from an abstract interface onto published service interfaces in WSDL, and returned results are properly presented to the user by the tool.

Once registry results are displayed, the user should select a proper candidate. Then, the tool performs three steps to adapt service interfaces and assemble internal components to it. The first step refers to build a proxy for the service. Proxy construction is automatically carried out by the tool. Then, the tool tries to build an adapter to map the interface of the proxy onto the abstract interface internal components expect. Finally, the tool indicates the DI container how to assemble internal components and adapters. Figure 3 summarizes the steps that are needed to proxy, adapt, and inject services.

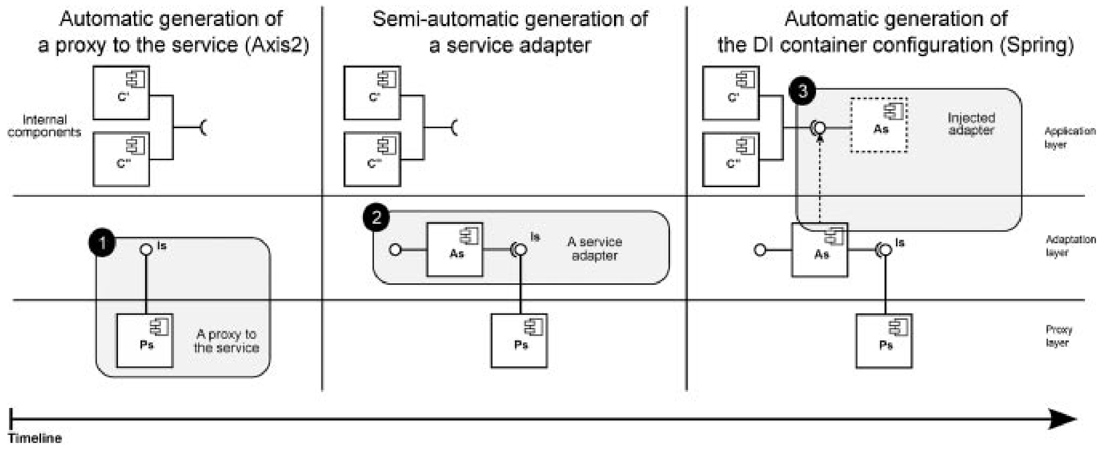

Automatic generation of a proxy to the service (Axis2)
Semi-automatic generation of a service adapter
Automatic generation of the DI container configuration (Spring)
Internal components
C'
C''
1
Is
Ps
A proxy to the service
2
As
Is
A service adapter
Ps
3
As
Injected adapter
As
Is
Ps
Application layer
Adaptation layer
Proxy layer
Timeline

Figure 3 Proxying, adapting, and injecting services with the EasySOC plug-in.

The current implementation of EasySOC employs Axis2 [25] for building service proxies and Spring [26] as the DI Container. Building a proxy with Axis2 involves giving as input the interface description of the target service (a WSDL document) to a command line tool. To setup the DI container, the names of internal components and services must be written in an XML file. For adapting external service interfaces to the internal abstract ones, we have designed an algorithm based on the work published in Ref. 27.

Our algorithm takes two Java interfaces as input (i.e., an abstract and an actual service interface) and returns the Java code of a service adapter. This adapter code commonly contains sentences relying on Java type castings to adapt the data types of the two interfaces. To do this, it starts by detecting to which operations of one interface the operations offered by the other should be mapped. The algorithm assesses operations similarity by comparing their names, documentation, and data types and names of their arguments. Data types similarity is based on a pre-defined similarity table that assigns similarity values to pairs of simple data types. Similarity between two complex data types is calculated in a recursive way. Once a pair of operations has been chosen, service adapter code is generated. To do this, the algorithm adapts simple data types by taking advantage of type hierarchies and performing explicit conversions. Complex data types are resolved recursively as well. Clearly, not all available mismatches can be covered by the algorithm. Therefore, users should revise the generated adaptation code, which makes this step semi-automatic.

As explained throughout this section, the plug-in presented performs the specific steps needed to design and implement service-oriented applications in accordance with the EasySOC development model. Besides guiding developers to produce better service-oriented applications in terms of maintainability, while accelerating the discovery process, the EasySOC plug-in aims at abstracting students from the problems and challenges that represent understanding the SOC paradigm and the plethora of technologies surrounding it, namely UDDI, WSDL, SOAP, and the distributed plus heterogeneous nature of SOC applications. Concretely, the approach to discover external services by automatically building queries and retrieving candidate services connects students to either UDDI or any UDDI-like service registry painlessly. Moreover, the distribution and platform heterogeneity are two concerns that students transparently deal with by employing proxy objects, which are built by EasySOC once a student has selected a candidate service. At the time of invoking a remote service, such proxies convert class messages into SOAP messages and transport them over the corresponding communication channel. Collaterally, such proxies provide a specification of external service libraries, but in students' preferred programming language instead of in WSDL and its data type system (i.e., the XSD language). Evaluation of EasySOC Section presents an evaluation of to which extent the EasySOC model and plug-in allow for a better level of abstraction and technology isolation when designing and implementing service-oriented software.

Using EasySOC: Step-by-Step Example

To understand the implications of modeling SOC applications with EasySOC, this section describes the design of a service-based personal agenda. The personal agenda is in charge of managing a contact list, arranging new meetings, and to notify these contacts of new planned meetings. The contact list is modeled as a collection of records with information about individuals such as name, current address (city, state, country, zip code, etc.), telephones, email addresses, etc. For the sake of clarity, we have simplified the functionality for coordinating the meeting by assuming that the participants being notified always agree with the arrangement provided by the requesting user.

Below we list the activities carried out by the personal agenda upon the creation of a new meeting. The text in italics represents the functionalities that will be not implemented but delegated to Web Services. We assume that the user of the personal agenda provides the date, time, participants, and location of the meeting upon its arrangement. Algorithmically, creating a new meeting roughly involves:

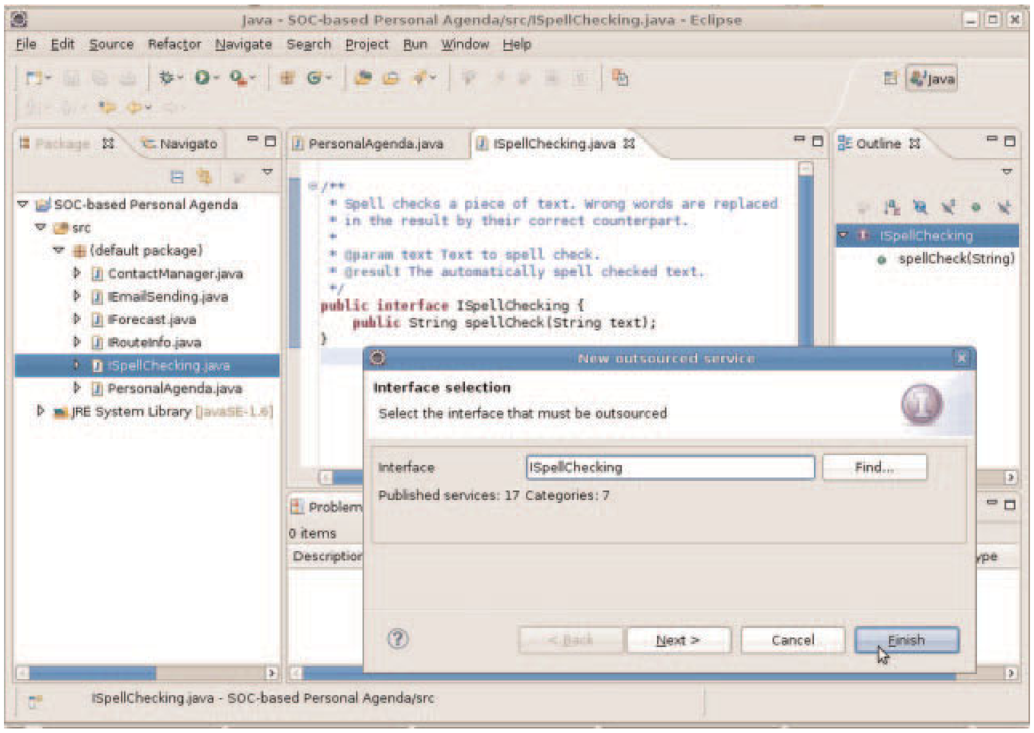
1. *Getting a weather forecast* for the meeting place at the desired date and time.
2. *Obtaining the routes* (or driving directions) that each contact participating in the meeting could employ to travel from their current address to the meeting place.
3. For each participant of the meeting:
   a. Creating an email with an appropriate subject, and a body including the weather report and the obtained route information.
   b. *Spell checking* the text of the email.
   c. Sending the email.

To build the above application, we start by designing its internal components. First, we define an internal component called PersonalAgenda, which is at the heart of the application and is in charge of coordinating the various services necessary for arranging a new meeting, and a ContactManager component representing the contact list. Then, we define four abstract interfaces used by the PersonalAgenda component, namely:

- IForecast: Returns a weather report for a given ZIP code.
- IRouteInfo: Supplies driving directions for a given source and target locations.
- ISpellChecking: Detects spelling mistakes in a given text.
- IEmailSending: Sends an email using a given body text and address.

At this point, our application consists of two internal components and four abstract interfaces, as listed in the left part of Figure 4. Then, we employ the EasySOC tool support for discovering services that provide a concrete implementation for the functionality modeled by these abstract interfaces. For instance, the next abstract interface Java code is used as a query when looking for spell checking services:

Figure 4 depicts the GUI of our plug-in within the Eclipse IDE. When discovering and then associating concrete Web Services instances to an abstract interface, users simply indicate through a dialog of our own such interface. The dialog shows how many services and categories are available in the registry about to be queried. Users are allowed to perform advances searches, for example, looking for services within an individual category. Finally, after querying the registry, a candidate list is presented to the user, as shown in Figure 5 (bottom). For each candidate Web Service, the offered operations are shown. Users can further browse and visualize the arguments and results of each operation in Java as well as WSDL format.



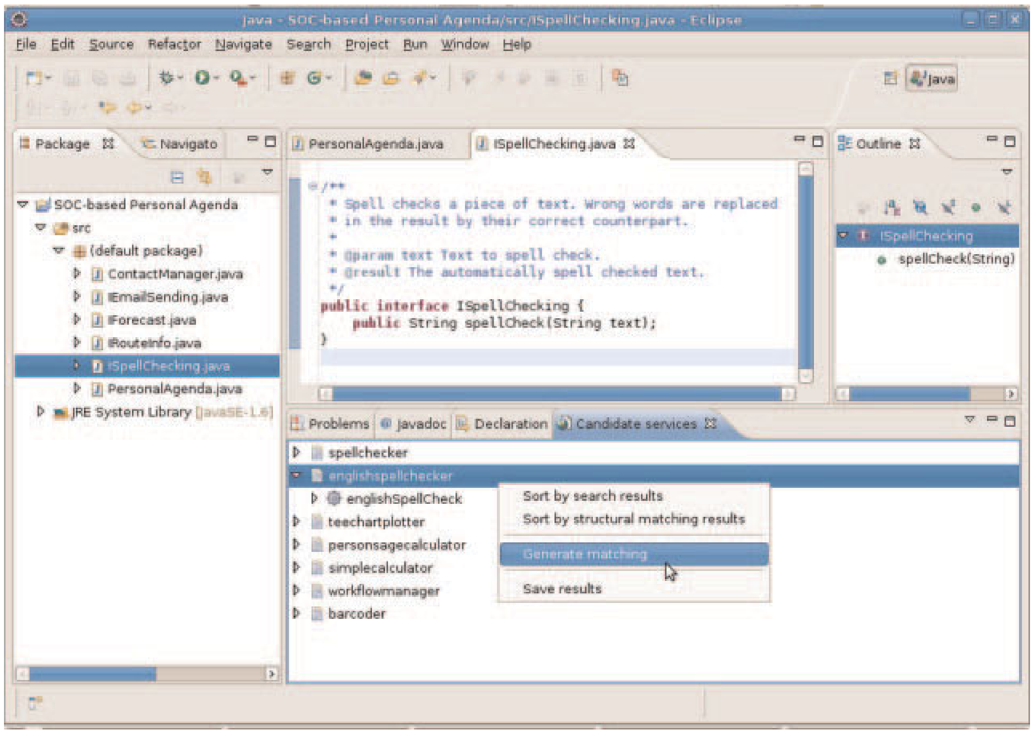
```
/**
 * Spell checks a piece of text. Wrong words are replaced
 * in the result by their correct counterpart.
 *
 * @param text Text to spell check.
 * @result The automatically spell checked text.
 */
public interface ISpellChecking {
    public String spellCheck(String text);
}
```

Select the interface that must be outsourced

Interface: ISpellChecking

Published services: 17 Categories: 7

Figure 4 The EasySOC plug-in: Discovering Web Services. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]



```
/**
 * Spell checks a piece of text. Wrong words are replaced
 * in the result by their correct counterpart.
 *
 * @param text Text to spell check.
 * @result The automatically spell checked text.
 */
public interface ISpellChecking {
    public String spellCheck(String text);
}
```

spellchecker
englishspellchecker
englishSpellCheck
teechartplotter
personsagecalculator
simplecalculator
workflowmanager
barcoder

Figure 5 The EasySOC plug-in: Selecting candidate Web Services. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

In our example, we have obtained two candidates services labeled "spellchecker" and a more specific "englishspell-checker." Let us suppose we select the second candidate, thus we must command our tool to build the corresponding proxy and adapter, and assemble them to the PersonalAgenda component. This is done via a contextual menu from the selected service (Figure 5). Afterwards, we revise the generated adapter code to ensure that all signature and data type mappings are properly coded and specified. The generated extra code artifacts will appear in a separate "mappings" folder of the project in the left part of the GUI.

Overall, the discovery–selection–injection sequence is performed until all external components of the application are associated with a service. To conclude, it is worth noting that user intervention was only required to select candidate services and revise adapters. Since an adapter indirectly interacts with a service through a proxy using the built-in object-oriented mechanism of method invocation, the user was free from dealing with WSDL, SOAP, and other SOC-specific technologies in the code of the application. Figure 6 illustrates the resulting application by remarking which activities take place during design and implementation time and which are supported by the tool.

EVALUATION OF EasySOC

This section describes the experiments that were performed to assess whether EasySOC, which supports the new methodology for constructing SOC applications described in the previous section, has an acceptable difficulty of adoption by novice users. Another aspect we evaluated is whether our tool allows for a better level of abstraction and technology isolation when designing and implementing service-oriented software compared to existing SOC libraries. The experiments involved 45 students and a two-phase homework, after which the students were asked to complete a survey to collect their opinions. Then, we analyzed these opinions to determine to what extent EasySOC helped them with the assignments.

The experiments were carried out during 2009 in the context of the "SOC" course (http://www.exa.unicen.edu.ar/∼cmateos/cos) of the Systems Engineering BSc program at the Faculty of Exact Sciences (Department of Computer Science—UNICEN). The course was also offered on 2008, is optional, and its audience are last-year undergraduate students and postgraduate students (both master and doctoral programs) without knowledge on SOC. The course requirements are good programming skills, object-oriented programming basics, and some experience with Java development. In 2009, the course was taken by 38 undergraduate students and 7 postgraduate students from 4 different Universities of Argentina.

The homework was carried out individually by the students, and each part of the work impacted on the partial and final grades for the course. This contributed to obtain a high level of commitment with the evaluation from students. As the experiment involved the use of a tool of our own, which might represent a threat to validity, the students were not told about the secondary goal of the homework, and precise and careful question–answering instructions prior to take the survey were emailed to them to ensure objectivity. After five lectures within 1 week of 2 h each discussing the fundamentals of the SOC paradigm and its enabling technologies, the students were instructed to develop the service-based personal agenda described in the Using EasySOC: Step-by-Step Example Subsection. The contents of the lectures comprised traditional SOC

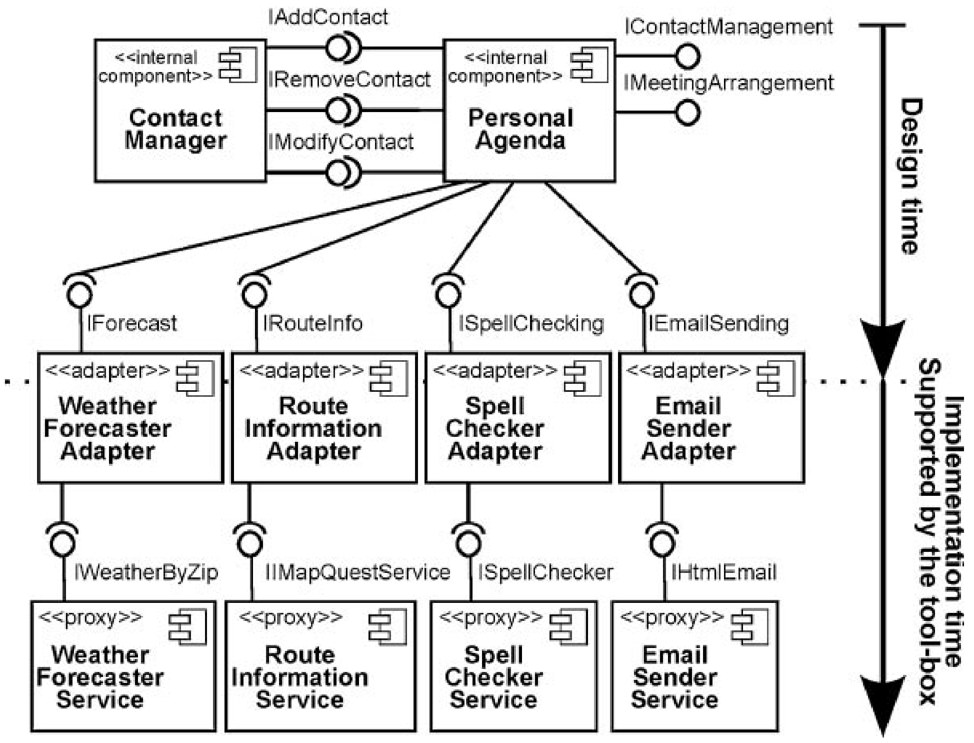


Figure 6 Component diagram of the service-oriented personal agenda.

Table 1 Results Based on 38 Undergraduate Students (UGS) and 7 Postgraduate Students (PGS)

Query item	Totally agree	Agree	Somewhat agree	Somewhat disagree	Disagree	Totally disagree
I would always design any SOC application as in the first phase						
UGS	1 (3%)	5 (13%)	18 (47%)	7 (18%)	6 (16%)	1 (3%)
PGS	0 (0%)	1 (14%)	2 (29%)	1 (14%)	2 (29%)	1 (14%)
I would always design any SOC application as in the second phase						
UGS	1 (3%)	16 (42%)	15 (39%)	3 (8%)	2 (5%)	1 (3%)
PGS	0 (0%)	5 (71%)	1 (14%)	0 (0%)	1 (14%)	0 (0%)
EasySOC materializes the triad SOC model						
UGS	1 (3%)	9 (24%)	14 (37%)	3 (8%)	0 (0%)	2 (5%)
PGS	3 (43%)	4 (57%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
EasySOC abstracts from Web Service technologies						
UGS	1 (3%)	14 (37%)	6 (16%)	1 (3%)	0 (0%)	0 (0%)
PGS	5 (71%)	2 (28%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
EasySOC simplifies service discovery						
UGS	27 (71%)	9 (24%)	1 (3%)	1 (3%)	0 (0%)	0 (0%)
PGS	5 (71%)	2 (28%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
EasySOC helps in changing service providers						
UGS	18 (47%)	11 (29%)	8 (21%)	1 (3%)	0 (0%)	0 (0%)
PGS	6 (86%)	1 (14%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

technologies and EasySOC. Among others, the set of traditional technologies described in the lectures included the W3C language for describing Web Services—i.e., WSDL, a popular library for invoking services within Java applications named Axis2, and an Integrated Development Environment that is designed for building Web-based and SOC-based applications using Java, named Eclipse WTP (http://www.eclipse.org/webtools).

As mentioned, the development of the software involved two phases. The second assignment was given after finishing the first one. In the first phase, the students designed the agenda software by using traditional Web Service libraries from the set of alternatives discussed in the lectures of the course[1] except EasySOC. Basically, these technologies were needed to invoke and incorporate selected services into their applications. In the second phase, the students developed the same application but by using EasySOC. It is worth noting that the order in which the two phases were performed did not bias the experiment in favor of any approach, as even when the same application was developed, students were familiar with the application domain prior to realize the two situations.

In both phases, the students exercised three aspects inherent to developing SOC applications, namely:

1. *Service discovery*: In the first phase, this was carried out by inspecting a UDDI Web Service registry by using its standard ‘‘Google-like’’ GUI that supports keyword-based search of Web Services. In the second phase, this was performed by using the Web Service discovery support of EasySOC.
2. *Service incorporation*: In the first phase, this involved building service proxies based on the service invocation features of the Web Service technology individually chosen by each student, whereas in the second phase this was uniformly handled by using the DI-based proxy and adaptation facilities of EasySOC.
3. *Service replacement*: The input service registry pointed to several implementations of the Web Services needed to develop the agenda software. The students were asked to change the provider for a half of the outsourced services *after* implementing their agenda software. For both phases, this involved repeatedly perform (1) followed by (2) on the already implemented software.

To better prepare the students to fill out the survey, we added some general ‘‘warming up’’ questions placed at the beginning of the survey, asking, for example, what SOC is and what kinds of applications actually benefit from it. Then, we included several query items designed to collect the students' opinions with respect to the three aspects mentioned above. By following Likert's approach to build questionnaires [28], the items were not plain questions but statements to which the students could either totally agree, agree, somewhat agree, somewhat disagree, disagree, or totally disagree. In this sense, students did not felt evaluated but consulted. Unlike other recent students' preference studies that have used an odd-numbered scale of agreement (e.g., [29,30]), we decided to employ an even-numbered scale to better capture the opinions of the students (no neutral mid-point). Additionally, students had to provide a concise but complete textual justification for each item. We also reserved a check box to indicate the perceived overall difficulty of the course and its assignments, and a text field through which any further comments could be specified.

Given the different formation levels of the students involved in the experiment, the next two subsections analyze the results by considering the opinions of the postgraduate students and the undergraduate students, respectively. Table 1 summarizes the survey query items (warming up questions have been omitted) and results. Query items were arranged in two groups, that is, those asking whether students would use either approaches for materializing service-oriented applications beyond this experience (items 1–2), and those evaluating the suitability of EasySOC according to supporting and simplifying the aspects that are inherent to SOC development (items 3–6). For the sake of better readability, the acronym PGS is used for referring to postgraduate students, while UGS represents undergraduate students.

[1]From now on, we will refer to employing the libraries and tools of this phase as the ‘‘traditional approach’’ to SOC-based design and implementation.

Postgraduate Students: Survey Analysis

For the first group of items, none of the surveyed postgraduate students completely agreed to always using either approaches for building their service-oriented applications, as shown in Table 1. However, 85% of the students either agreed or somewhat agreed to the idea of ''using EasySOC in early stages of development,'' since the pattern-based programming model of EasySOC could lead to some adaptation effort when SOC-enabling existing applications to made them compliant to the EasySOC application anatomy. However, the same students said that they would definitively use the tool in the presence of large service registries whose functional content is not known regardless of the development stage. This is precisely the case of open contemporary massively distributed environments such as the Web or Grids, in which a lot of services are offered. As the number of publicly available services grows, it is crucial to have effective and efficient discovery mechanisms to dramatically narrow down the result list and therefore reduce the effort when looking for required services [24].

Furthermore, one student disagreed with always using EasySOC because he/she through that its discovery mechanism would not be effective when dealing with poorly described WSDL documents (the same student consistently disagreed with not employing any other invocation library in those cases when a lot of services are available). Although not particularly relevant to the goals of our experiment, these are correct observations, on which we have been working on by identifying common anti-patterns in WSDL descriptions that harm our service discovery mechanisms and providing clear user guidelines to avoid them [31]. To complement this research with evidence taken from a cognitive perspective, we conducted an extra and optional survey among the students to gather opinions about which WSDL construction practices they felt are more detrimental to understanding what a service does, which in turn difficult service selection. For details on this study, please see Appendix. We are therefore planning to incorporate these ideas based on the feedback from students in the near future in order to improve our tool support regarding service ranking and selection.

Returning to the Likert-based questionnaire results, four out of the seven students disagreed with different levels to using the Web Service libraries employed in the first phase of the homework because such libraries demanded them to spend much time rewriting the application upon changing service providers, introducing complexity to the assignment and leaving less time to invest into SOC-specific design issues. In other words, they thought that having an adaptation layer for isolating application code from service interfaces supports the construction of service-oriented software in a more technology-agnostic way. The other three students said that they would rely on the traditional approach to service consumption as long as the set of services to be consumed is known in advance, that is, service instances are given as input to the assignment. However, these three students consistently responded that they would switch to EasySOC in those situations when target services are not determined beforehand, such as collaborative homework in which students play different roles from the Web Service model, as some support for service discovery would then be strongly necessary.

On the other hand, for the second group of items, all postgraduate students either totally agreed or agreed to the associated query items. Most of them said that EasySOC provides intuitive support to the triad find-invoke-publish when working with Web Services, even when they did not exercise the ''publish'' activity in the homework but nevertheless acknowledged wizard-based tool support for it. Certainly, materializing such model directly in a software tool allows students to focus on performing the activities that correspond to high-level SOC design. Moreover, the students considered that EasySOC was useful to make them unaware of technological details with regard to finding or consuming Web Services. Concretely, half of the students conceived inspecting service registries and providing code for processing WSDL descriptions as being the most time-consuming and difficult tasks when constructing their SOC applications. One student pointed out, however, that even when abstraction from technological details is important, so is to have background on low-level technologies for those cases in which specific adjustments must be made to an application (e.g., changing the invocation protocol used to call an individual service). In this sense, EasySOC automatically generates the necessary technology-dependent software artifacts for contacting external Web Services, while allows such artifacts to be inspected and modified by users when necessary.

The seven postgraduate students found the service discovery module of EasySOC ''very helpful to quickly find required candidate Web Services,'' which essentially means that looking for Web Services implementing the functionality a user application expects is efficient and hence has a positive impact on application building in terms of required effort. Furthermore, four out of the seven students found that good code documentation in their client-side software artifacts was a prerequisite for the discovery process of EasySOC to be effective. Indeed, the effectiveness in finding required services heavily depends on to what extent users employ explanatory names and proper documentation for both class names and method parameters. However, these are desirable and frequent development practices that should be followed in any kind of software [32] that require little cognitive effort provided the application domain is known and thus good documentation for its implementation code can be supplied. Finally, all of the postgraduate students said that EasySOC helped them with service replacement, which arguably may translate into a cleaner apprehension of this SOC-specific concept.

Undergraduate Students: Survey Analysis

Table 1 shows that, for the case of undergraduate students, the opinions with respect to items 1 and 2, and to a lesser extent for the items 3–6, were less concentrated as opposed to the results of the previous subsection. In this sense, to better analyze the responses, we quantified and categorized whether each individual student was more convinced of using an SOC approach above the other. Thus, for example, if a student *agreed* to ''I would always design any SOC application as in the first phase'' and *somewhat agreed* to ''I would always design any SOC application as in the second phase,'' it meant that the student preferred the traditional approach. Figure 7 illustrates the obtained results. It is worth pointing out that, except for the case of the ''Undecided'' group, the rest of the students either *somewhat agreed*, *agreed*, or *totally agreed* to one of these two items, which established a minimum acceptable level of confidence regarding approach preference.

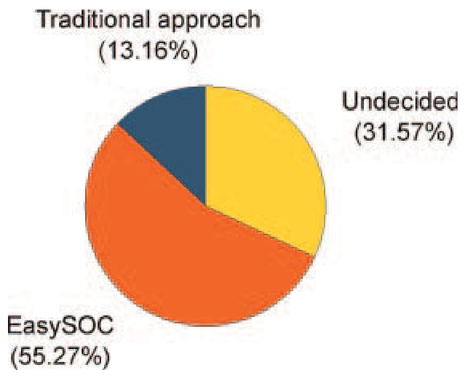


Figure 7 Undergraduate students: Approach preference. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Remarkably, 55.27% of the surveyed students said that they preferred using EasySOC over relying on the traditional approach. The common argument behind this preference was that the basic elements of the EasySOC methodology facilitate the "agile" construction of "modifiable" SOC applications. Agility in this context comes as a result of allowing users to organize the components of their applications according to the SOC paradigm without the requirement of unnecessarily dealing with low-level SOC technologies. On the other hand, modifiability aligns with the benefits of EasySOC in terms of structuring components to better visualize the points of an application that are potentially affected by the concept of service replacement. In this line, the students emphasized on the usefulness of the discovery support and the convenience of the automatic source code generation techniques for building service adapters of the EasySOC plug-in.

Furthermore, 5 out of the 38 students (13.16%) said that they were more comfortable with the traditional approach since it required less software (just a proxy library to invoke services) compared to EasySOC, and "one could nevertheless achieve an acceptable level of decoupling and abstraction between applications and service interfaces by addressing this non-functional requirement early in the design stage of the application" to simplify code maintenance and service replacement. Precisely, EasySOC comes with a software support that prescribes a simple SOC development methodology that is based on popular object-oriented patterns, which leads to a natural way of building SOC applications and therefore performing the associated paradigm shift. Application design is thus more focused on specifying the functionality of the internal application components and the external services without initially paying attention to technological details, which allows the user to concentrate on exercising the fundamental elements of SOC design.

Not surprisingly, 31.57% of the undergraduate students were not decided about which approach they would use to develop SOC applications in the future. Moreover, half of them (i.e., 6 students) simultaneously *somewhat agreed* to using both tools because "choosing a tool depends on several factors," including the size of the client-side software, the number of services to be consumed, and the amount of dependencies between internal application components and such services. However, the same students pointed out that they found EasySOC useful to simplify service discovery, and to keep the client source code away from "service-specific instructions," which allowed them to be more focused on SOC design and simplified the requirement of changing service providers.

On the other hand, the other half of the students gave origin to two corner cases. Three students agreed to employ either approaches since they had trouble learning Eclipse but they would definitively exploit the design principles materialized by EasySOC for doing SOC. As explained, these principles promote technology-agnosticism, and we are in fact working on providing alternative materializations of EasySOC for supporting other popular DI containers and programming environments to avoid this "platform-barrier" given by the supporting technologies on which the current implementation of EasySOC relies. Furthermore, two students and one student simultaneously *disagreed* and *completely disagreed*, respectively, on using either alternative for developing applications for similar reasons. They nevertheless gave encouraging values to the query items 3–6 in favor of EasySOC accordingly.

All Students: Acceptance Analysis

In order to obtain complementary quantitative evidence on the opinions of all the students participating in the experiment, the Likert scale [28], the most widely used psychometric scale in survey research, was employed. Roughly, the Likert scale is the sum of answers on several Likert items, that is, individual statements to which respondents can associate a level of agreement. After a survey is completed, the agreement levels of each Likert item are typically summed to create an overall score per participant.

Since we were interested in quantifying the overall perception of the students on EasySOC, we associated a numerical score with query item 1 ranging from 0 (totally agree) to 5 (totally disagree), but ranging from 5 (totally agree) to 0 (totally disagree) for query items 2–6. As a consequence, our designed Likert scale was in the range of [0,30], with 0 being strongly disagree with EasySOC and with 30 being strongly agree with it. We computed the Likert score per student. Figure 8 shows the score histogram, where each bar contains the number of students who had the same score. Interestingly, only one participant got the lowest score (15), that is, the worst perception was in the middle of the entire scale.

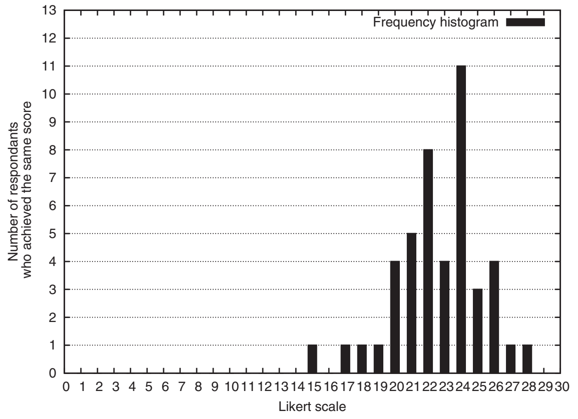


Figure 8 Likert scale: Frequency of the scores.

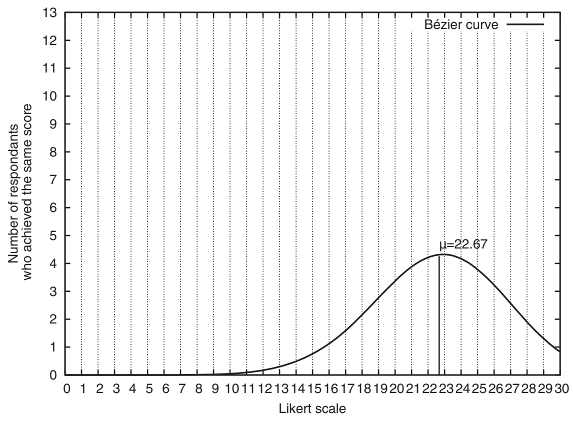


Figure 9 Likert scale: Distribution of the scores.

Figure 9 shows that by smoothing these results using Bézier curves, they tended to a normal distribution with an average $\mu = 22.67$ and a standard deviation $\sigma = 2.65$. Then, 95.4% of the students scored between $[\mu - 2 \times \sigma, \mu + 2 \times \sigma]$. In other words, 42 students scored in the range of [17.36, 27.97], which manifests a very good perception of EasySOC from the experience.

CONCLUSIONS

Service-oriented computing is a relatively new paradigm for developing applications that promotes the seamless combination and reuse of existing pieces of functionality exposed by third parties. The paradigm is far from being a buzzword and is being exploited in the software industry by means of specialized libraries for both exposing and invoking services. Consequently, there is an increasing need of effective ways of teaching the fundamentals of this revolutionary paradigm to systems engineering students.

One of the biggest hurdles in the path of educating students in SOC concepts is the plethora of technologies surrounding the paradigm, which often obscures its cornerstones and eclipses its simplicity. To date, very few educational tools for SOC have been proposed, which unfortunately capture a small fraction of the essential aspects of the paradigm. In addition, the benefits of using these approaches have not been experimentally assessed in real learning situations yet. Thus, technology isolation at the correct level of abstraction seems to be a fundamental precondition to rapidly convey the basic concepts of SOC and alleviating the cognitive effort that the associated paradigm shift unavoidably demands.

To help addressing these issues, in this article we have described the EasySOC tool, which allows users to easily design and build service-oriented applications. EasySOC enforces the usage of common object-oriented design patterns and component-based notions to structure such applications in an effort to bridge the gap between these older paradigms and SOC. Furthermore, EasySOC simplifies service discovery and invocation, and hides technological details from users, which allows them to be more focused on exercising and manipulating the basic elements of SOC-based design. We evaluated the educational benefits of EasySOC by investigating whether exploiting students' experience in such earlier and well-established programming paradigms while facilitating activities inherent to SOC and hiding technologies reduces the effort needed to effectively start applying SOC design concepts. We argue that, even when assuming that some knowledge on object-oriented programming from prospective users is necessary might represent a threat to applicability of our approach, it is known that object orientation is ubiquitous since it is already present in a very large percentage of Computer Science and Information Systems academic programs [33].

Our study involved 45 computer science students, who were asked to develop a service-oriented application by using traditional SOC libraries and EasySOC. Then, their thoughts were collected via a Likert-based questionnaire. Results show that the students, who had no previous experience in SOC development before the situation, perceived that EasySOC allows focusing on essential aspects of the paradigm while leaving secondary ones on background. This suggests that EasySOC is a convenient tool for having the first encounter with SOC concepts and at the same time developing real applications. Since the students had conventional programming skills and knowledge before taking our SOC course, which is the state of advanced system engineering students in most universities, we can reasonably extrapolate these results to support the argument that EasySOC may be useful as a tool support for other similar courses.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments and suggestions to improve the article. We also thank the students who participated in the survey for their good predisposition in the experiment. Finally, we acknowledge the financial support provided by ANPCyT through grant PAE-PICT 2007-02311.

APPENDIX:ALL STUDENTS: SELECTION PREFERENCES ANALYSIS

Certainly, service discovery is an essential aspect of the SOC paradigm. As users have the final word on which service is more appropriate to their purposes when using EasySOC, we conducted another experiment with all the students to analyze the reasons behind selecting among several candidates, as illustrated in Figure 5. As input, we provided them with different WSDL documents and an extra questionnaire that consisted of 11 questions divided into 3 groups. A group of questions were designed to simply familiarize the participants with each WSDL document. For example, one question asked about the number of operations offered by the WSDL documents. The second group of questions asked the students about whether the WSDL documents were self-explanatory enough so they understand what the offered service does from a functional perspective and how to invoke it, or if their descriptiveness could be improved to some extent instead. The last group of questions allowed the participants to comment which version of the employed WSDL documents would they prefer and why. The questions of the second and third groups, and the main results of this extra survey are described next.

First, we gave the students a WSDL document with several operations belonging to the same application domain, but one operation in a different domain, and in turn asked the students whether removing this non-cohesive operation would improve the clarity and understandability of the service or not. A 92% of the students answered that they would remove the non-cohesive operation.

Second, we gave the students a service operation that returns a generic data type, and whose documentation provides hints that, in case of an invocation or execution problem, error information would be included in the output message returned by the service. Then, we asked the students about whether they could determine the structure of the operation response, whether they would replace the data type of the operation output with a data type that merely represents the operation result, and if they preferred the WSDL document to include error information within output messages or provide separate messages to convey such information. A 92% of the students answered that the structure of the output was not clear at all. The rest of the participants answered that the analyzed operation always returns instances of xsd:double or xsd:int data types, that is, the WSDL data types that represent doubles and integers, respectively. This result may stem from the fact that the operation was for uploading data files, and if a file is successfully transferred via the service, then the stored file size is returned. In this sense, it seems that 8% of the students disregarded the possibility of a failure during the execution of the operation. The results also showed that 92% of the students would replace the data type of the output of the operation. As the reader can see, the percentage of participants that identified the situation as a problem was exactly the same that voted for replacing the data type definition. At the same time, 92% of the students realized that the analyzed output message could exchange error information. However, only 81% of them answered that they would use a WSDL built-in error message to separate return error information.

Finally, we gave the students a WSDL document with two operations returning the same data types, but defined twice. The students were asked whether they would remove one of the redundant data types or not. An 81% of them answered that they would remove the repeated data types, because "it obscures the data types defined by the WSDL operations of the service."

The last group of questions allowed the participants to comment which one of the input WSDL documents they would select and why. The comments made by the students provided an idea of their preferences when selecting WSDL documents. Some participants included two, or more, different preferences in their comments. From these comments we summarized and ranked the most frequent students' preferences. Accordingly, the identified preferences are listed below in decreasing order of occurrence:

1. The data types exchanged and exposed by the operations of the selected WSDL document were better represented.
2. The selected WSDL document was more concise.
3. The operations of the selected WSDL document belonged to a single application domain.
4. Error information was better handled by the selected WSDL document.

Specifically, the results showed that 37% of the students included in their comments the reason related to better representation of data type definitions. The responses of 30% of the

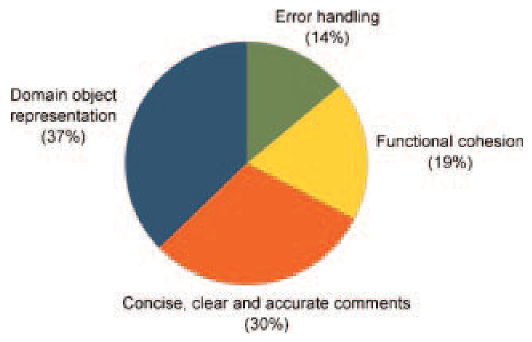


Figure 10 Students' criteria when selecting among several candidate services. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

students highlighted that they preferred concise WSDL documents. Besides, 19% of the students commented that they would select those WSDL documents that arranged cohesive operations. Finally, 14% of the students said that separating error information from output messages helped them to understand how to use the service, so they would preferably select a WSDL that deal with error information in this manner. Figure 10 summarizes these results.

REFERENCES

[1] M. Bichler and K.-J. Lin, Service-oriented computing, IEEE Comput 39 (2006), 99–101.
[2] J. Erickson and K. Siau, Web Service, service-oriented computing, and service-oriented architecture: Separating hype from reality, J Database Manage 19 (2008), 42–54.
[3] W.-T. Tsai, Y. Chen, C. Cheng, X. Sun, G. Bitter, and M. White, An introductory course on service-oriented computing for high schools, J Inform Technol Educ 7 (2008), 315–338.
[4] B. Lim, C. Jong, and P. Mahatanankoon, On integrating Web Services from the ground up into CS1/CS2, SIGCSE Bull 37 (2005), 241–245.
[5] W.-T. Tsai, Y. Chen, and X. Sun, Designing a service-oriented computing course for high schools. In: S. C. Cheung, Y. Li, K.-M. Chao, M. Younas, and J.-Y. Chung (Eds.), IEEE International Conference on e-Business Engineering (ICEBE 2007), IEEE Computer Society, Hong Kong, China, 2007, pp 686–693.
[6] C. Kelleher and R. Pausch, Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers, ACM Comput Surv 37 (2005), 83–137.
[7] M. Kendall and E. Gehringer, Teaching Web Services with Water. In: Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference, San Diego, CA, IEEE Computer Society, Los Alamitos, CA, 2006, pp 7–12.
[8] J. Nandigam, V. Gudivada, and M. El-Said, Teaching Web Services using WSExplorer. In: Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference, Milwaukee, WI, IEEE Computer Society, Los Alamitos, CA, 2007, pp S3H-20–S3H-25.
[9] W.-H. Wu, W.-F. Chen, L.-C. Fang, and C.-W. Lu, Development and evaluation of Web Service-based interactive and simulated learning environment for computer numerical control, Comput Appl Eng Educ 18 (2009), 407–422.
[10] W.-H. Wu, W.-F. Chen, T.-L. Wang, and T.-J. Su, A pedagogical Web Service-based interactive learning environment for a digital filter design course: An evolutionary approach, Comput Appl Eng Educ 18 (2010), 423–433.
[11] B. Ramamurthy, GridFoRCE: A comprehensive resource kit for teaching Grid Computing, IEEE Trans Educ 50 (2010), 10–16.

[12] I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organization, Int J High Perform Comput Appl 15 (2001), 200–222.

[13] S. Vaughan-Nichols, Web Services: Beyond the hype, Computer 35 (2002), 18–21.

[14] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, The next step in Web Services, Commun ACM 46 (2003), 29–34.

[15] Google, Inc., Google SOAP search API, 2006, http://code.google.com/apis/soapsearch/reference.html.

[16] C. Depcik and D. Assanis, Graphical user interfaces in an engineering educational environment, Comput Appl Eng Educ 13 (2005), 48–59.

[17] J. García Perez-Schofield, F. Ortín Soler, E. García Roselló, and M. Pérez Cota, Towards an object-oriented programming system for education, Comput Appl Eng Educ 14 (2006), 243–255.

[18] G. Licea, J. Reyes Juárez, L. Martínez, and L. Aguilar, Developing programming tools to reach a deeper understanding of advanced programming concepts, Comput Appl Eng Educ 16 (2008), 305–314.

[19] B. García Perez-Schofield, E. García Roselló, F. Ortín Soler, and M. Pérez Cota, Visual Zero: A persistent and interactive object-oriented programming environment, J Visual Lang Comput 19 (2008), 380–398.

[20] M. Crasso, C. Mateos, A. Zunino, and M. Campo, EasySOC: Making Web Service outsourcing easier, Inform Sci (2010), in press.

[21] C. Mateos, M. Crasso, A. Zunino, and M. Campo, Separation of concerns in service-oriented applications based on pervasive design patterns. In: Proceedings of the 2010 Web Technology Track (WT)—ACM Symposium on Applied computing (SAC), Sierre, Switzerland, ACM Press, New York, NY, 2010, pp 2509–2513.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley, Reading, MA, 1995.

[23] M. Crasso, C. Mateos, A. Zunino, and M. Campo, Empirically assessing the impact of dependency injection on the development of Web Service applications, J Web Eng 9 (2010), 66–94.

[24] M. Crasso, A. Zunino, and M. Campo, Easy Web Service discovery: A Query-By-Example approach, Sci Comput Program 71 (2008), 144–164.

[25] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana, and G. Daniels, Axis2, middleware for next generation Web Services. In: Proceedings of the IEEE International Conference on Web Services, IEEE Computer Society, Los Alamitos, CA, 2006, pp 833–840.

[26] C. Walls and R. Breidenbach, Spring in action, Manning, Greenwich, CT, 2005.

[27] E. Stroulia and Y. Wang, Structural and semantic matching for assessing Web Service similarity, Int J Coop Inform Syst 14 (2005), 407–438.

[28] R. Likert, A technique for the measurement of attitudes, Arch Psychol 22 (1932), 1–55.

[29] A. Savoy, R. Proctor, and G. Salvendy, Information retention from PowerPoint^{TM} and traditional lectures, Comput Educ 52 (2009), 858–867.

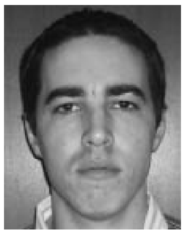
[30] W. Woody and C. Baker, E-books or textbooks: Students prefer textbooks, Comput Educ 55 (2010), 945–948.

[31] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, Improving Web Service descriptions for effective service discovery, Sci Comput Program 75 (2010), 1001–1021.

[32] D. Spinellis, The way we program, IEEE Softw 25 (2008), 89–91.

[33] D. Douglas and B. Hardgrave, Object-oriented curricula in academic programs, Commun ACM 43 (2000), 249–256.

BIOGRAPHIES



Cristian Mateos (http://www.exa.unicen.edu.ar/∼cmateos) received a Ph.D. degree in Computer Science from the UNICEN, in 2008, and his M.Sc. in Systems Engineering in 2005. He is a full time Teacher Assistant at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, Grid middlewares and Service-oriented Computing.

Marco Crasso (http://www.exa.unicen.edu.ar/∼mcrasso) received a Ph.D. degree in Computer Science from the UNICEN in 2010. He is a member of the ISISTAN and the CONICET. His research interests include Web Service discovery and programming models for SOA.

Alejandro Zunino (http://www.exa.unicen.edu.ar/∼azunino) received a Ph.D. degree in Computer Science from the UNICEN, in 2003, and his M.Sc. in Systems Engineering in 2000. He is a full Adjunct Professor at UNICEN and member of the ISISTAN and the CONICET. His research areas are Grid computing, Service-oriented computing, Semantic Web Services and mobile agents.

Marcelo Campo (http://www.exa.unicen.edu.ar/∼mcampo) received a Ph.D. degree in Computer Science from the Universidade Federal do Rio Grande do Sul, Brazil, in 1997. He is a full Associate Professor at the UNICEN, Head of the ISISTAN, and member of the CONICET. His research interests include intelligent aided software engineering, software architecture and frameworks, agent technology and software visualization.